NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USING THE AGILE DEVELOPMENT METHODOLOGY AND APPLYING BEST PRACTICE PROJECT MANAGEMENT PROCESSES

by

Gary R. King

December 2014

Thesis Advisor:	John S. Osmundson
Second Reader:	Daniel P. Burns

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2014	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE USING THE AGILE DEVELOPMENT METHODOLOGY AND APPLYING BEST PRACTICE PROJECT MANAGEMENT PROCESSES	5. FUNDING NUMBERS
6. AUTHOR(S) King, Gary R.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

There are many software development life-cycle models. Each one has its own advantages and disadvantages, forcing program management to select carefully before embarking on a full-scale development effort. A popular choice today is the Agile development model, due to its more informal processes and ability to adapt easily to changes. However, one of these positive elements is also one of its negative aspects. These less formal processes can lead developers to use the Agile model as authorization to avoid any process efforts, leading to a difficult project management problem.

This thesis explores the manner by which the Agile development model may be executed in a disciplined manner. The thesis also describes the application of various techniques to create a robust development environment while still maintaining the value the methodology brings. In addition, it also highlights the importance of selecting each practice carefully and applying that practice uniquely to each project to ensure maximum performance.

14. SUBJECT TERMS Agile Software Development Life cycle, Project management, requirements management, system architecture, risk management, test, documentation	15. NUMBER OF PAGES 99
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

USING THE AGILE DEVELOPMENT METHODOLOGY AND APPLYING BEST PRACTICE PROJECT MANAGEMENT PROCESSES

Gary R. King
Civilian, Department of the Navy
B.S., Texas A&M University Corpus Christi, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
December 2014

Author: Gary R. King

Approved by: John S. Osmundson
Thesis Advisor

Daniel P. Burns
Second Reader

Clifford A. Whitcomb
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

There are many software development life-cycle models. Each one has its own advantages and disadvantages, forcing program management to select carefully before embarking on a full-scale development effort. A popular choice today is the Agile development model, due to its more informal processes and ability to adapt easily to changes. However, one of these positive elements is also one of its negative aspects. These less formal processes can lead developers to use the Agile model as authorization to avoid any process efforts, leading to a difficult project management problem.

This thesis explores the manner by which the Agile development model may be executed in a disciplined manner. The thesis also describes the application of various techniques to create a robust development environment while still maintaining the value the methodology brings. In addition, it also highlights the importance of selecting each practice carefully and applying that practice uniquely to each project to ensure maximum performance.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ADL	Architectural Description Language
BUFD	big up-front design
CMMI	Capability Maturity Model Integration
DoDAF	Department of Defense Architecture Framework
DSDM	dynamic software development method
EMV	expected monetary value
FDD	feature driven development
QA	quality assurance
RBS	risk breakdown structure
SDLC	software development life cycle
SWOT	strengths, weaknesses, opportunities, threats
TDD	test driven development
TOGAF	The Open Group Architecture Framework
UI	user interface
UML	Unified Modeling Language
YAGNI	you ain’t gonna need it

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

As the software development world evolves, so do the development methodologies used to apply discipline to the task. According to the research organization Forrester Research International, one of the more popular software development life cycles by a significant margin today is the Agile development model (West and Grant 2010, 2–3). The Agile model is a lightweight methodology definition offering adaptable processes with early and frequent iterative product deliveries. However, the basis of the model as defined by The Agile Manifesto specifically deemphasizes some traditional project management elements (2001). This thesis explores the application of project management processes in the Agile development methodology. It uncovers numerous opportunities for application of defined processes, but also reveals that the appropriate level of rigor is defined uniquely by each project's Agile implementation.

One of the phases essential in any project management model is requirements gathering and management. Requirements define the basis of the system, so a well-defined and easily followed mechanism should be defined. In fact, requirements management is so important that the primary reason for failure of projects is poor execution of this process, according to the Standish Group (1995). The Agile method uses a different approach than traditional methods by working through requirements as the project progresses, rather than defining and securing the requirements at the beginning. This approach turns the "Iron Triangle" upside down by focusing the project on requirements as the variable element of the task rather than time and funds existing as the variable aspects (Leffingwell 2011, 17). In this way, the Agile model offers a great deal more flexibility as the project progresses and new requirements are introduced and defined. However, in order for a project to support project requirements that are not defined and fixed at the beginning of the project, this demands high customer interaction and involvement to continue the defining and prioritization actions throughout the project duration. This issue can be alleviated using various Agile ceremonies such as daily meetings, frequent reviews, and demonstrations. Another important aspect setting Agile apart in requirements management is the fact that rather than having requirements teams

gather and document the requirements independently and then provide those requirements to the development team in the next phase of the process, all members of the development team are involved in requirements discovery documenting these requirement findings in user stories. This requirements management method increases exposure for all team members and maintains focus on the product definition.

Another vital aspect to a system's development is a well-defined architecture. Like requirements management, Agile applies focus on this effort not only at the beginning of the project, but as the project evolves. While critics argue that not defining the entire architecture at the inception of the project will result in the misinterpreting requirements, others argue this tactic is the only way to guarantee the right product is delivered. The latter argument defends the position suggesting that as the project evolves, so should the architecture. While the system architect may lead this effort on an Agile team, the responsibility for this work resides with the entire team. Again, this increases the team's knowledge and awareness, as well as positively affecting the acceptance of the result by the entire development team. The Agile model offers several methods to support this approach including applying a defined discovery process starting before the actual project begins, creating user stories (requirements/tasks) for the system architect that must be planned and executed, and incorporating the expected results into the Agile team's "Definition of Done" (Cohn 2013).

Unlike the first two project management practices reviewed in this thesis, many of the defined Agile processes inherently assist the risk management practice. This is accomplished via many elements include the prioritized backlog that allows riskier requirements to be addressed early, the ability to change direction as the needs of the Stakeholders evolve or are better understood, smaller incremental planning cycle allowing for more accuracy, early delivery and testing cycles forcing early identification of issues, among others. However, because of the interpretation of some project managers of the Agile Manifesto as a methodology focusing only on rapid delivery eliminating time consuming project management practices, the system architecture task is often ignored. This results in the responsibility for applying risk management is with the stakeholders who must make it a priority. From that point, all members of the Agile

development team will be forced to be involved in the process, and there are many methods that can be used. These include both qualitative and quantitative means. From the qualitative perspective, the project may choose to perform an exhaustive review of the backlog selecting the highest risk items and then prepare a risk task board or preparing an evaluation of the project from various perspectives and preparing a risk breakdown structure (RBS) annotating each from multiple perspectives such as probability, impact. On the quantitative side, projects may choose to take the RBS one step farther applying values to each area and preparing decision trees that depict the project's best path or use other mathematical methods that arithmetically determines the riskiest requirements providing the project a measureable decision tool for requirements prioritization.

The last elements addressed in this thesis are testing and documentation. These types of quality assurance practices are often seen in the Agile methodology definition as opportunity for avoidance and time savings. However, depending on the project, these may be just as important as delivery of the actual product. Testing in the Agile paradigm is iteratively executed with each incremental delivery. This creates an opportunity for defects to be recognized early in the development process, but since all members of the Agile development team have a role in testing there is concern in the independent aspect of the test execution. Testing early also forces the practice to be executed in smaller increments rather than being executed at the end and potentially being curtailed due to time constraints. For this reason, Agile projects accentuate unit tests to a much higher degree than other models. This forces automated testing to be defined and executed providing better overall code coverage. The test driven development (TDD) practice is often used to assist in this approach. This tactic has the developer define the tests based on acceptance criteria defined by the stakeholder before the code is developed. This ensures that the stakeholder will be delivered the requested functionality, as well as ensuring complete understanding by the development team of the requirement.

Depending on the project, documentation may be a vital deliverable; however, attention is not often applied to this practice. Like other Agile requirements, stakeholders and project members must apply importance to this practice, defining it in relation to project success. All members are responsible for the documentation of the project from

varied perspectives to ensure all required artifacts from specifications to code are successfully completed. This can best be accomplished by creating a defined documentation pattern for a project to follow, which encompasses all of the most important aspects for the targeted audience.

The Agile development methodology has shown opportunities for delivery of increased business value. But, the implementation of the model also offers the opportunity to ignore valuable project management practices. This thesis shows that while beginning with the Agile Manifesto, these definitions are lacking, the implementation of these practices are abundant and should be implemented at the discretion of each project manager. There exists a wealth of successful, robust methods easily adaptable to any Agile project. The challenge is ensuring this lightweight approach delivers the required functionality while at the same time ensuring priority is applied to the individual project-appropriate disciplines.

LIST OF REFERENCES

Agile Manifesto. 2001. "History: The Agile Manifesto." Accessed May 4, 2014. http://agilemanifesto.org/history.html.

Cohn, Mike. 2013. "Clarifying the Relationship between Definition of Done and Conditions of Satisfaction." *Mike Cohn's Blog*. August 21. http://www.mountaingoatsoftware.com/blog/clarifying-the-relationship-between-definition-of-done-and-conditions-of-sa.

Leffingwell, Dean. 2011. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Boston, MA: Pearson.

Standish Group International, Inc. 1995. *The Standish Group Report: CHAOS*. Accessed May 26, 2014. http://www.projectsmart.co.uk/docs/chaos-report.pdf.

West, Dave and Tom Grant. 2010. *Agile Development: Mainstream Adoption Has Changed Agility for Application Development & Program Management Professional*. City, MA: Forrester Research, Inc.

ACKNOWLEDGMENTS

First, I would like to extend my appreciation to Dr. John Osmundson for reading and reviewing draft after draft of chapters and data throughout the last year and providing valuable feedback to aid in completing the thesis.

In addition, I would like to thank my wife, Kimberly, and children, Lauren and Rachel. Your patience and support through the course curriculum and thesis development process were a tremendous source of comfort and encouragement to me. I could not have done it without you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The Agile development model continues to gain popularity over traditional development models as it provides a more lightweight method of software development while also helping more projects to be successful. Figure 1 is taken from a blog maintained by Mike Cohn, who is considered one of the fathers of Agile. The figure shows the results of the Standish Group's evaluation of projects executed between 2002 and 2010. It depicts the success rates of projects executing the Waterfall model versus those implementing Agile. The second graphic, Figure 2, is extracted from VersionOne—an Agile tools provider. It portrays the value proposition this methodology offers. Higher visibility through project execution, more adaptability, the capacity to deliver more business value early, and the benefit of driving down risk are several reasons for the methodology's growing popularity.

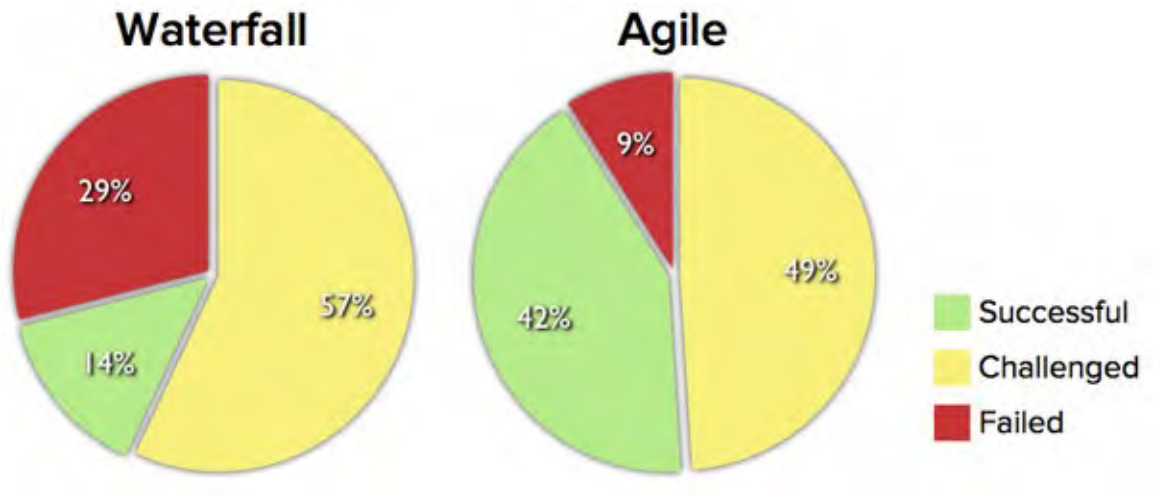

Figure 1. Project Success Rates (from Cohn 2012)

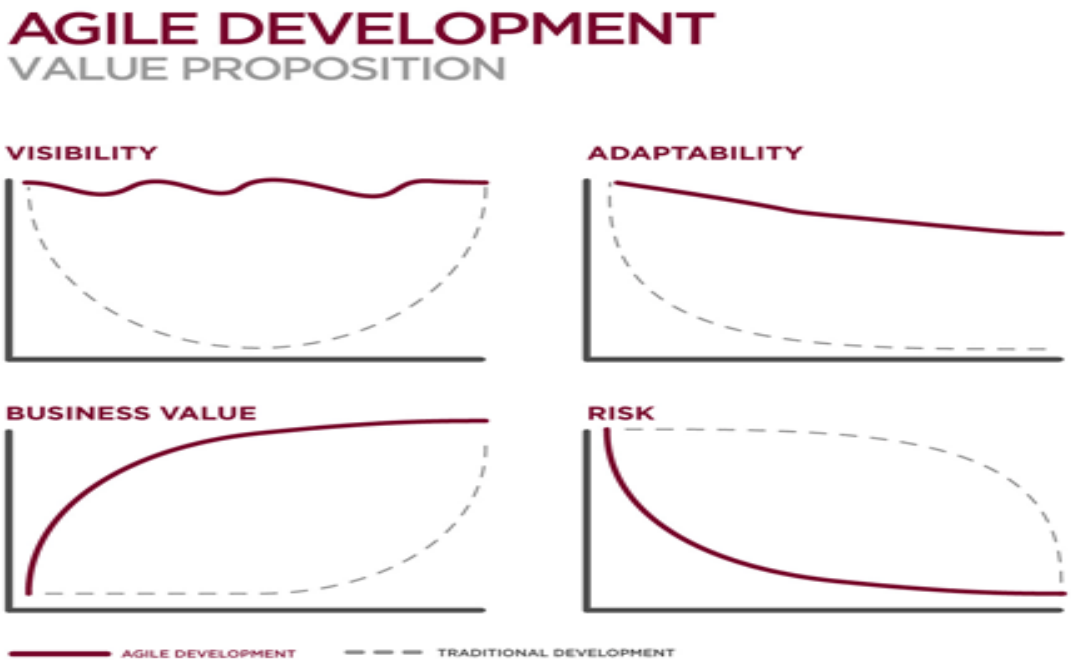

Figure 2. Agile Value Proposition [x-axis is time] (from `2013)

While this difference in success rate and value has led many organizations to adopt the methodology, it is not without its detractors. One of the disadvantages identified in some executions of Agile is its inherent lack of documentation and process. While advocates suggest the approach allows for the "right" amount of documentation and process as determined by the project members, others suggest that it is simply eliminating essential steps of project management.

B. PURPOSE

The purpose of this thesis is to investigate the application of project management processes in the Agile development methodology.

C. RESEARCH QUESTIONS

The following question and sub-questions will be addressed:

- What is the best practice for incorporating requirements management, risk management, system architecture definition, and testing/documentation into the Agile development methodology?
    - What was the genesis of Agile and the rationale for its rapid adoption?
    - What are the varied Agile development implementations?
    - Do any of the development implementations lend themselves to application of project management practices?
    - Does application of the project management practices ultimately create a better product or provide no real value to the Agile development life cycle?

D. BENEFITS OF STUDY

The result of this research and analysis is to evaluate the application of the Agile development model when considering its use on a project. Depending on the project, this evaluation will assist decision makers in determining whether Agile or some other methodology that may be best suited thereby improving both the product at delivery and in sustainment.

E. SCOPE AND METHODOLOGY

This thesis will first describe the Agile model to establish a baseline of knowledge for the reader. It does not attempt to cover every aspect of software project management. Rather, it reviews four distinct areas of the discipline to provide a cross section view for the reader. This will be followed by presenting both sides of the Agile conflict in the application of varied project management practices as it specifically investigates Agile implementation in the areas of requirements management, systems architecture, risk management, and testing/documentation. Finally, the thesis provides a conclusion with recommendations for best practices.

THIS PAGE INTENTIONALLY LEFT BLANK.

II. THE AGILE DEVELOPMENT MODEL DEFINED

A. INTRODUCTION

The chapter provides a brief background of the advent of the Agile development methodology along with a short, illustrative description of one type of Agile implementation and a comparison to the more traditional Waterfall model.

B. GENESIS OF THE AGILE DEVELOPMENT MODEL

The Agile development model was conceived by a small group of individuals interested in lean development. This diverse group included contributors from Extreme Programming, Scrum, Dynamic Software Development Method (DSDM), Adaptive Software Development, Crystal, Feature Driven Development (FDD), and Pragmatic Programming (Agile Manifesto 2001). The group convened in Utah in 2001 to discuss these various methodologies and to agree on a philosophy that puts into practice the saying that people are a company's greatest assets. The result of this summit was the Agile Manifesto, which was created to outline the collective values of the group. The Agile Manifesto (2001) reads:

> We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:
>
> *Individuals and interactions* over processes and tools
> *Working software* over comprehensive documentation
> *Customer collaboration* over contract negotiation
> *Responding to change* over following a plan
>
> That is, while there is value in the items on
> the right, we ***value the items on the left more***.

This document was authored and subsequently signed by the 17 participants in the meeting and has since been signed by thousands of believers via the organization's web page - http://agilemanifesto.org/history.html.

C. AGILE DEVELOPMENT METHODOLOGIES DEFINED

One specific Agile development methodology is called Scrum. The name and idea came from Hirotaka Takeuchi and Ikujiro Nonaka in a 1986 article, in which the authors compared traditional development models to "relay races" as opposed to a more "holistic or 'rugby' approach—where a team tries to go the distance as a unit, passing the ball back and forth" (1986, 137).

From this simple concept a revolution began. Many methods for applying the Agile methodology have been defined and are in practice today. Using Takeuchi and Nonaka's idea, Scrum is a process developed by Ken Schwaber and Jeff Sutherland that is an iterative approach to software development. It maintains a very limited definition because the process attempts to apply the ultimate amount of flexibility to the team and the project. Figure 3 provides a pictorial of the Scrum development process.

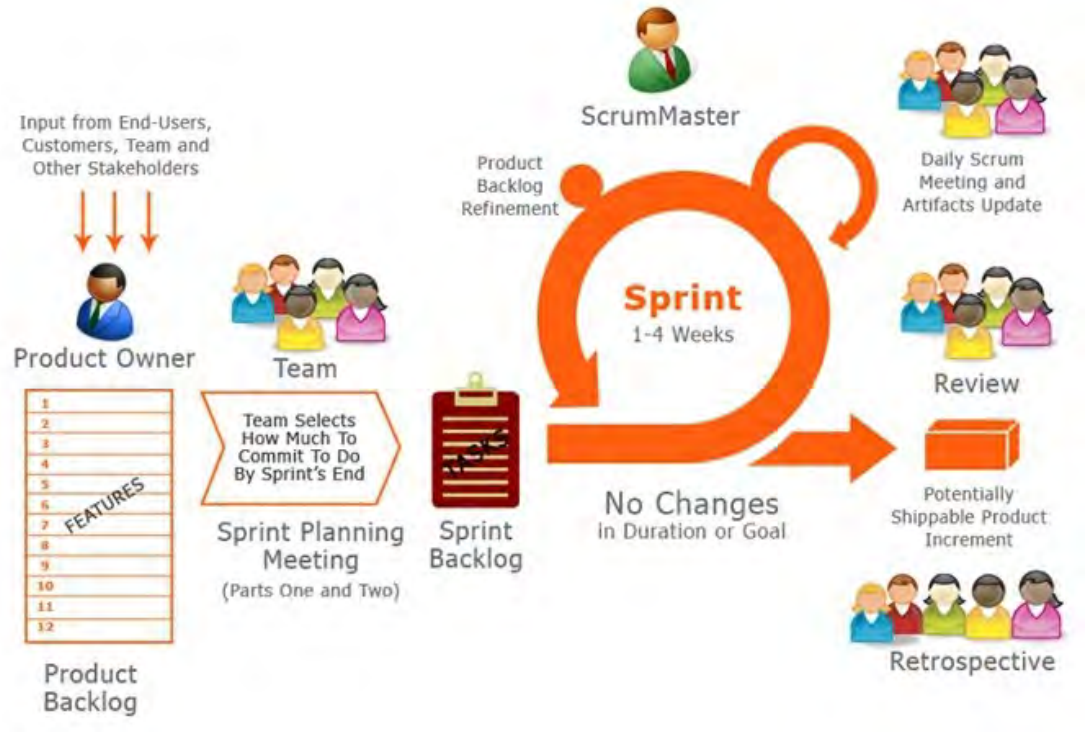

Figure 3. Scrum Model (from N-Axis Software Technologies 2010)

Once a vision is identified and the product backlog is filled with user stories—Agile's version of requirements—which are defined and prioritized, the scrum team executes an iterative development cycle, called a sprint, within a 30-day timeframe. The result of the sprint is a product that is reviewed with the product owner for feedback. New items are then selected from the product backlog and the cycle starts again. This sequence provides an opportunity for earlier and more valuable feedback from the product owner than exist in traditional process models. At the beginning of a project, given only a 30-day turnaround in the Scrum model, these responses are more easily applied to a tangible product than to a plan or drawing, which is often the artifact from traditional models after such a short duration. Scrum is defined by its inventors as an empirical process, and Schwaber points to three important elements of control that need to be applied to any process of this sort: visibility, inspection, and adaption (2004, 3). Schwaber states that all elements of a project need to be visible to ensure those controlling the process understand the state of the project at any given time. He identifies inspection to ensure that the process is evaluated on a frequent basis to ensure undesirable practices are identified and adjustments can be incorporated into the process. Schwaber also identifies adaption, suggesting that changes to the process are valuable and important if they are determined to improve the product. Each of these elements will be referenced throughout the sections below as the incorporation of system architecture into the Scrum model is discussed.

It is important to note that when applying a process-oriented approach to any project, the implementers must be careful in employing the appropriate model. Agile practitioners argue that application of the wrong approach can suppress the creativity of the team, but without an adequate process in place, it may be difficult to accomplish the goals of the project.

D. CONTRAST TO WATERFALL DEVELOPMENT METHODOLOGY

The waterfall development model is an organized, methodical process for developing software. It is referred to as the waterfall because a specific set of sequential steps are followed, and it is often represented in a descending manner much like Figure 4.

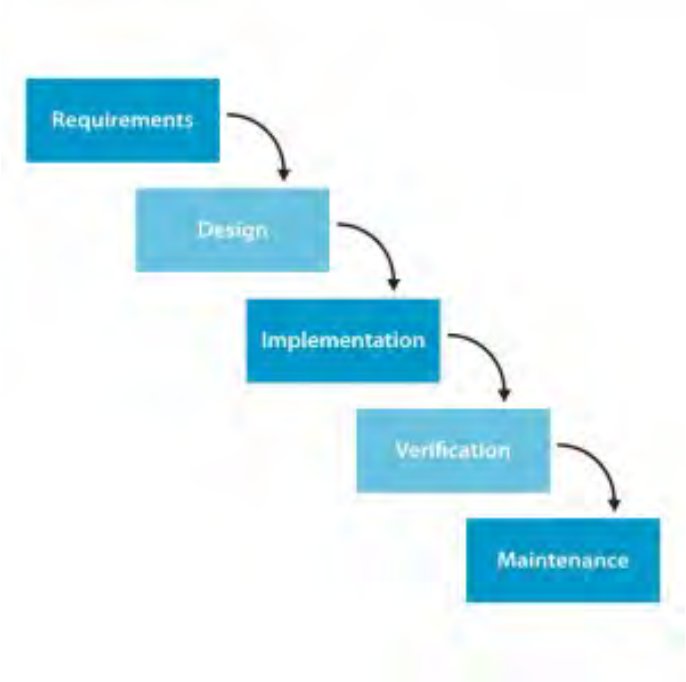

Figure 4. Waterfall Model (from Waterfall Model 2014)

This methodology helped bring structure to a software discipline that one author called “haphazard integrated software network like cluttered knitting” (Waterfall Model 2014). The phases of the model are defined to assist the development team in executing the software development in a structured set of steps, completing each phase before beginning the next, and using the results of the previous stage to drive the process of the next stage. Below is a brief description of each phase (DotNetBlocks 2011).

- Requirements—gathering and defining requirements to be implemented during the project.
- Design—defining the system from an architectural and software perspective based on the requirements collected during the requirements phase.
- Implementation—development of the code in satisfying the defined requirements and in accordance with the defined design.
- Verification—testing the system from end-to-end to ensure all requirements have been met.
- Maintenance—passing the finished system to the customer to validate along with remedying any defects found.

While this model has met with tremendous success over time, the inability to return to a phase after completion, difficulty in perfectly gathering requirements, failure to allow the customer to change requirements after initial definition, and the pace of development have all been drawbacks (DotNetBlocks 2011). These issues led one author comparing the waterfall and Agile methodologies to state:

> The Waterfall method is incredibly rigid and inflexible. Altering the project design at any stage in the project can be a total nightmare and once a stage has been completed, it is nearly impossible to make changes to it. In addition, the problem with the Waterfall method is that feedback and testing are deferred until very late into the project. So if there is a problem, it is very difficult to respond to it, requiring a substantial amount of time, effort, and sometimes money. (Mikoluk 2013)

E. CHAPTER SUMMARY

The Agile model was borne from the Agile Manifesto in 2001 and has achieved enormous popularity since its inception. The Agile Scrum method is one application of the Agile model that focuses on short sprint cycles and high customer involvement. The waterfall model has lost popularity to the Agile model in terms of use, but has maintained a solid methodology historically used by practitioners. However, Agile has extended some of its popularity in recent years due to its ability to address some of the shortfalls of the waterfall model.

THIS PAGE INTENTIONALLY LEFT BLANK

III. REQUIREMENTS MANAGEMENT

A. INTRODUCTION

Arguably, one of the most important phases in project management is requirements gathering and management. Requirements for a systems development represent the basis of the effort, and from this the system is described in detail. Regardless of the development methodology executed, without a well-defined process including defined methods for requirements gathering, documenting, processing through customer vetting techniques, maintaining configuration management, and managing validation through the testing phase of development, projects will have extreme difficulty succeeding. According to the Standish Group's CHAOS report based on 8,380 applications, the primary reasons projects fail are due to various factors (Figure 5), but requirements management and a shortage of customer interaction rank the highest (Standish Group International, 1995, 9). One author places blame on both the stakeholders of the system and the developers. Ellen Gottesdiener (2002) in her book *Requirements by Collaboration* explains that developers repeatedly ask for clarification on requirements but are often stuck interfacing with business people rather that users. Conversely, developers are often too focused on "the newest technologies, tools, and methods" rather than on the solution itself. (Gottesdiener 2002, 5). Gottesdiener goes on to write that this "leads to a focus on *building the product right* rather than *building the right product*" (2002, 5).

This section will describe the Agile Scrum method for requirements management identifying the processes and issues for developing a system focusing specifically a defined process and the need for teamwork between all parties.

Project Impaired Factors	% of Responses
1. Incomplete Requirements	13.1%
2. Lack of User Involvement	12.4%
3. Lack of Resources	10.6%
4. Unrealistic Expectations	9.9%
5. Lack of Executive Support	9.3%
6. Changing Requirements & Specifications	8.7%
7. Lack of Planning	8.1%
8. Didn't Need It Any Longer	7.5%
9. Lack of IT Management	6.2%
10. Technology Illiteracy	4.3%
Other	9.9%

Figure 5. Project Impaired Factors (from Standish Group International, Inc. 1995, 9)

B. APPLYING REQUIREMENTS MANAGEMENT TO AGILE SCRUM

Each methodology defines some sort of requirements management practice, and Agile Scrum is no different. The following sections describe varying approaches to accomplish this important project practice.

1. Challenges and Opportunities

As Dean Leffingwell (2011, 16) states in his book, *Agile Software Requirements*, "We [practitioners] take a far more flexible approach to requirements management: one that is temporal, interactive, and just in-time." (In this context, the author is referring specifically to Figure 6. While the traditional development models fixed requirements and allowed resources and delivery timeframes to be flexible, Agile takes the opposite approach, identifying requirements as the variable, flexible component and resources and delivery dates fixed.

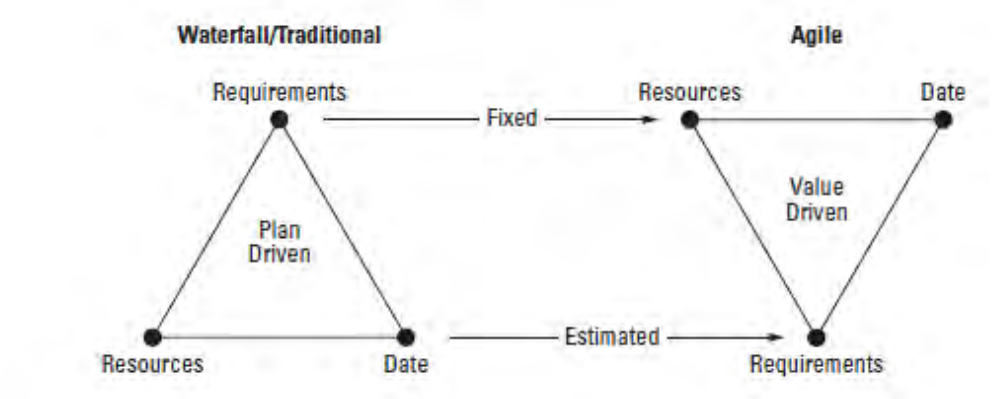

Figure 6. Agile Iron Triangle (from Leffingwell 2011, 17)

While this represents a potential shift in priority, it also creates challenges to the development team, specifically in the form of stakeholder involvement. All development models require customer interaction. However, in the traditional case, requirements are often composed in the beginning of the project, and then identified as the foundation for cost and schedule (Leffingwell 2011, 6). These requirements are then adjusted as the development team begins analysis, but with the cost and schedule already fixed the project may already have issues before it begins. This issue has been identified as a "root cause of project failure" (Leffingwell 2011, 7). In Agile, the requirements are discovered through various methods with direct stakeholder interaction. This level of collaboration is so coveted, the preference for development teams is to have the customer on-site (Figure 7). The top bar chart shows 60 percent of all Agile programs have the customer on-site, which leads to a high satisfaction level. It is also important to note that there is a lower percentage of variability in requirements as well as lower percentage of requests for early product delivery, and zero percent unsatisfied clients. The study suggests these latter elements are a result of incremental deliveries that "appears to better satisfy customer needs" (Ceschi 2005, 25) as customers are able to see the product on a more consistent basis, realizing the result of their requirements input. There also exists a potential downside to this high stakeholder involvement in the form of the development of features developed outside of the requirements. To this end, the study suggests that in this model the "customer may refine and modify requirements" and it actually "encourages requirements changes" (Ceschi 2005, 25).

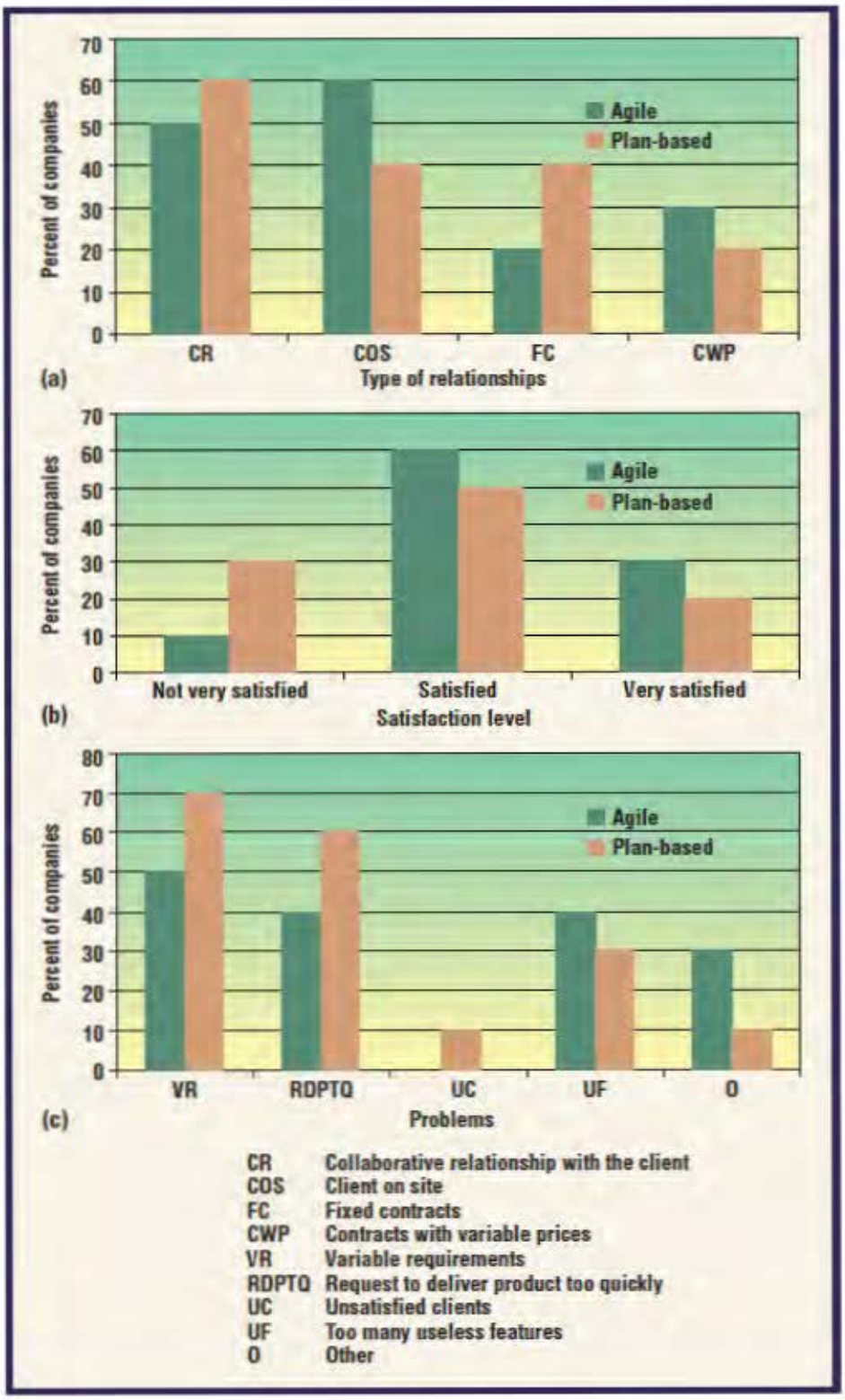

Figure 7. Customer Relationship (from Ceschi 2005, 25)

High customer interaction is imperative, but can evolve into a time burden for the development team. This potential issue may be overcome using various communication techniques. In one study, with both multiple distributed customer bases and development teams, the group held daily virtual meetings with the development teams, held a weekly gathering with the stakeholders, set up a wiki site for collaboration, as well as other tools such as "a bug management system, newsgroups, and email to document history and preserve discussion trails" (Young 2008, 305). A second case study of a global software development suggests the following Agile processes made cooperation easier:

- Daily Scrums – the most valuable of practices. This is a short meeting not to exceed 15 minutes in length held literally on a daily basis, where each member of the team answers three questions:
    - "What did you do since the scrum meeting?
    - Do you have any obstacles?
    - What will you do before next meeting?" (Paasivaara, Durasiewicz, and Lassenius 2009, 197—198).
    - The result of this meeting, according to the participants in the study was that the meeting, "…increased transparency to the other site, getting a good overview of what was happening in the project and adding communication across sites" (Paasivaara, Durasiewicz, and Lassenius 2009, 198).
- Weekly Scrum-of-Scrums—these meetings were also identified as being key to success. The participants were a different member of the various scrum teams each week along with the scrum masters. These meetings lasted 30 minutes and addressed the entire scrum team's activities. This was meant to be a more abstracted view of the work effort, so each representative answered the three questions from their respective team's perspective along with two more:
    - "Have you put some impediments in the other teams' way?
    - Do you plan to put any impediments in the other teams' way?" (Paasivaara, Durasiewicz, and Lassenius 2009, 198).
    - Although direct, these questions were meant to generate conversation about multiple team integration, and the practice was deemed successful. (Paasivaara, Durasiewicz, and Lassenius 2009, 198–199)
- Sprints—this served as the cycle through which work on the development task was accomplished. The important element of this process is with the short duration, well-defined expectations were created for each team and

member increasing transparency among the teams. (Paasivaara, Durasiewicz, and Lassenius 2009, 199).

- Sprint Planning Meetings—these meetings defined the sprint goals and were held immediately before the commencement of a sprint cycle. In this case study, the planning was divided among several separate meetings both onsite and at the distributed sites. These did not come without their challenges with technology (e.g., malfunctioning cameras and conferencing tools), but with each successive meeting more persons participated in estimating their own team's efforts, and created more "visibility to the work on both sites." (Paasivaara, Durasiewicz, and Lassenius 2009, 199–200).
- Sprint Demos—a ceremony in which the sprint capabilities are demonstrated to stakeholders and other interested individuals. This is another opportunity increase visibility for all stakeholders and scrum development teams. (Paasivaara, Durasiewicz, and Lassenius 2009, 200)
- Backlogs—listing of features to be developed on the system. These were made visible to all team members and updated daily giving full visibility into the project progress. (Paasivaara, Durasiewicz, and Lassenius 2009, 200)
- Frequent Visits and Multiple Communication Modes—the studies cited periodic "face-to-face meetings," "team building exercises," and "social events" allowed the teams to "…discuss difficult issues, and get a better picture of the project" (Paasivaara, Durasiewicz, and Lassenius 2009, 199–202). In addition, the study suggested all media were used for communication including "email, phone calls, chat, application sharing and teleconferencing." This plethora of tools was deemed essential depending on the communication required, quick question and answer, lengthy discussion, or technical dialogue.

2. Agile Team Responsibilities

From the description above it becomes clear that there are many persons involved in the requirements management process. In other development life cycles, only certain team members are involved in the development of requirements, but in the Agile methodology that is changed. Dean Leffingwell writes,

> The entire team is integrally involved in defining requirements, optimizing requirements and design trade-offs, implementing them, testing them, integrating them into a new baseline, and then seeing to it that they get delivered to the customers. That is the sole purpose of the team. (2011, 8)

Within Agile Scrum there are three main practitioners in the requirements process. These are the product owner, the Scrum master, and the team, consisting primarily of the developers and the testers. Each of these role owners has a specific function.

- Product Owner—this individual is the representative from the business community that is serving in a liaison position to the team. This person works to define the requirements, prioritizes the backlog for all aspects of the program, provides acceptance of the requirements as each is developed, and is the front door for new requirements as they arise. (Leffingwell 2011, 51)
- Scrum Master—this role is responsible for facilitating advancement towards completing requirements, helping the team with embracing continuous improvement, administering the implementation plan with the team, and removing impediments to the team's progress. (Leffingwell 2011, 52)
- Developers—these performers are responsible for writing the code that implements the requirements. However, this group is also responsible for evaluating and understanding the requirements, authoring and executing automated unit and acceptance tests for requirements evaluation. (Leffingwell 2011, 52)
- Testers—these individuals are working in conjunction with the Developers and have a virtually identical set of processes to follow, except from a test perspective. Testers are responsible for validating requirements are satisfied, but are also ensure understanding of acceptance criteria to author acceptance tests and assist in automation techniques for testing. purposes. (Leffingwell 2011, 53)
- Other roles the team will interface with are:
    - Architects—this role is discussed at length in Chapter IV.
    - Quality Assurance—these individuals ensure code quality and assist testers in validating completion of requirements. (Leffingwell 2011, 54)
    - Other Specialist and Supporting Personnel—any of the set of specialists that assist in product engineering and delivery. This may include, but is not limited to, configuration manager, build manager, interface designers, technical writers, and the like. (Leffingwell 2011, 54)

Figure 8 depicts the ideal Agile team with the repository resources on the left and the processes/individuals which each interface with on the right.

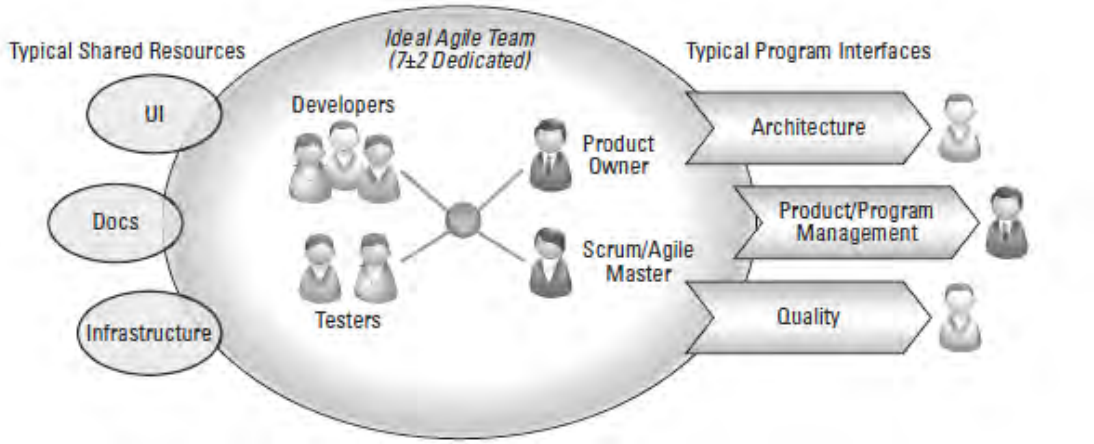

Figure 8. Ideal Agile Team Depiction (from Leffingwell 2011, 53)

3. Process Methods

The most common method for performing requirements management in Agile is maintaining requirements in user stories as briefly mentioned in Chapter II. Simply defined, "A user story is a brief statement of intent that describes something the system needs to do for the user" (Leffingwell 2011, 100). The communication used by user stories are made up of three elements—card, conversation, and confirmation (Leffingwell 2011, 102).

- Card—location where the user story is briefly documented. (Leffingwell 2011, 102)
- Conversation—embodies the exchange between members of the Agile Team to ensure understanding of the goal of the story. (Leffingwell 2011, 103)
- Confirmation—represents the manner by which the Stakeholders will validate the story has been successfully fulfilled. (Leffingwell 2011, 103)

A specific format is used for each story to define an understandable communication common to both the stakeholders and the developers. Taken directly from Leffingwell's book, it reads as follows:

As a <role>, I can <activity> so that <business value> where:

- <role> represents who is performing the action or perhaps one who is receiving the value from the activity. It may even be another system, if that is what is initiating the activity.

- <activity> represents the action to be performed by the system.
- <business value> represents the value achieved by the activity. (2011, 103).

While the "activity" signifies the requirement itself, the other two elements offer context that is often missing from traditional requirements methodologies. In conjunction with documenting the user story, the Agile Team will also document acceptance criteria. This is the "confirmation" element described previously, that identifies the measure by which the Stakeholder will be satisfied with the implementation. This is not unit or functional test criteria, but rather a simple definition from the user of recognizing successful completion (Leffingwell 2011, 105). Figure 9 provides a graphical representation of this process in which a *Story* (user story) is a *Backlog Item*, completed via a *Task*, and completed when the *Story Acceptance Test* is satisfied.

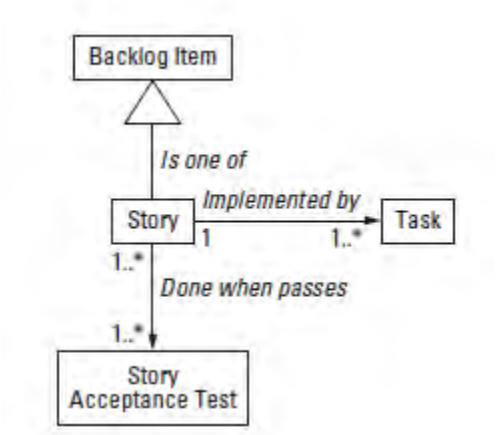

Figure 9. Requirements Model (from Leffingwell 2011, 99)

A collection of user stories that defines a system is termed a backlog. From an execution perspective, it is really just a "'to-do' list for the team" (Leffingwell 2011, 157). This backlog can take many forms in terms of documenting requirements for an entire system, a release, or a single build. While, any member of the Agile Team can write a user story, both the backlog prioritization and maintenance falls to the product owner (Leffingwell 2011, 55). Prioritization and maintenance incorporates the concept of a change to a requirements baseline, generating change management. This is also an

important aspect of requirements management; one area that Agile executes very well. Because the product owner is responsible for prioritization, each iteration—or sprint—may encounter a refinement or change to the requirements order. In either case, refinement or not, when the backlog is consulted for planning, the next story is extracted and planned for the current iteration (Leffingwell 2011, 159). The development team estimates the work to be performed within the given timeframe of the iteration/sprint (two to four weeks), and then commits to the effort (Leffingwell 2011, 160). This commitment may appear to resemble the Iron Triangle turned upside down in Figure 5, but has some distinct differences according to Leffingwell:

- The commitment is for the period of the iteration not for the period of the project, which is why each is kept short. (2011, 160).
- "This commitment is made by the teams, not for the teams" (2011, 160).
- The goals allow flexibility in implementation (2011, 160).

One method for achieving maintenance of requirements, or user stories, is via a Story Wall. This is either a physical or electronic wall that holds the system's requirement in user stories on index cards. It is important to note that index cards are referenced above and specifically here used to force the stories to be short, concise, and as simplistic as possible. Both the front and back may be used, but the representation is to keep the stories from becoming overly complex. The cards are then prioritized by the product owner and moved, literally moved to increase transparency, as the development moves between stages (Delgadillo and Gotel 2007, 377–378) and support the iterative process defined.

C. CHAPTER SUMMARY

Requirements management is often viewed as one of the most important phases of the software development life cycle. Many projects fail due to the inability to adequately define requirements for a project, but Agile offers some tools to assist. Agile turns the "Iron Triangle" upside down focusing on requirements as the variable element in the project which leads to opportunities for the customer to adjust over time. Many process approaches assist in this regard including scrums, sprints, and frequent demonstrations and communication with team members and the customer. All members of the Agile

development team have a responsibility in requirements management whether it is in the form of defining a user story / acceptance test definition, estimation, or consultation.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SYSTEMS ARCHITECTURE

A. INTRODUCTION

An essential element to any system's development project is a well-defined architecture. In software, this phase is as important as or more important than any other phase in the development life cycle. One author makes this assertion stating, "The right architecture paves the way for system success. The wrong architecture usually spells some form of disaster" (Klein 2008, 3). While there are many definitions of architecture, there are many more for systems and software architecture. In their book, *Software Architectures in Practice*, the authors define software architecture as, "…the structure or structures of the system, which comprise the software elements, the externally visible properties of those elements, and the relationships among them" (Bass, Clements, and Kazman 2003, 21). There are many approaches by which systems architecture may be defined and many development models and frameworks that may be employed. This chapter explores the Agile methodology and its approach to defining and applying architecture.

B. APPLYING SYSTEMS ARCHITECTURE TO AGILE SCRUM

This section reviews the various challenges and opportunities to implementing a defined architecture approach using Agile Scrum. It compares traditional methods to Agile and reviews various industry opinions, defines the responsibilities of the team members, and provides a number of implementation possibilities.

1. Challenges and Opportunities

While other models focus on system architecture at the beginning of a project and then revisit the plan over time as the project progresses, Agile Scrum completes a system architecture planning cycle with each sprint in 30-day progressions. There are pros and cons to this methodology.

Supporters argue that defining the architecture as the project evolves is the only way to ensure the right product is developed that will ultimately support the effort. "You

don’t know what you don’t know” is a familiar mantra with which these factions will identify, suggesting that an incremental, iterative architecture definition allows for facts to reveal themselves. As the project continues from sprint to sprint, more and more information is gathered by the development team. The farther along the project progresses, the more definitive and precise this information becomes. With this precision, a better architectural design can be created. The technical note, Capability Maturity Model Integration (CMMI) or Agile: Why Not Embrace Both!, prepared by the Software Engineering Institute at Carnegie Melon, summed it up best when stating, “The product and architectural components are not locked in so that developers can perform continuous trade-offs until all the component requirements are available” (Glazer et al., 2008, 23). An important element to recognize is that despite the architecture’s iterative nature, the system is always required to be operational at the conclusion of any sprint. Therefore, some aspects of the system will have to be decomposed and developed in logical sections and then developed to support the sprint deliverable itself. Figure 10 visually depicts this process showing the “Architectural Runway” supporting varied product features and evolving over time. In this diagram, time is depicted on the X-axis with architecture feature implementation on the Y-axis, showing that while there is always the focus on requirements to support the system’s long-term features, architectural features are incorporated iteratively to support current sprint deliveries and development.

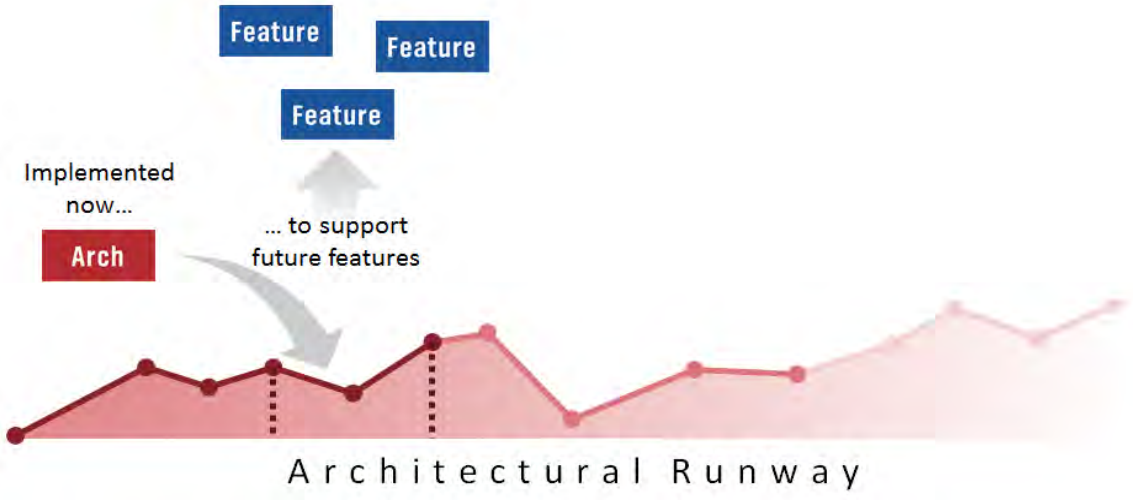

Figure 10. Evolution of System Architecture [x-axis is time] (from Yakyma and Leffingwell, 2013)

It is also important to acknowledge that not every change to the system from a developer perspective creates an architectural impact. In fact, according to one source, only a small set of decisions made by the programming group will ever be architecturally significant (Abrahamsson, Ali Babar, and Kruchten 2010, 18).

Agile factions have gone as far as to coin new acronyms to define the issues with other development methods, calling them antiquated, big up-front design (BUFD); they point at the analysis and documentation that accompanies it, stating it is generally "you ain't gonna need it" (YAGNI) (Abrahamsson, Ali Babar, and Kruchten 2010, 16).

Detractors from the methodology suggest that Agile practitioners have a false sense of predictability and misinterpret the requirements. The last line of the Agile Manifesto (2001) reads, "while there is value in the items on the right, we value the items on the left more." This line has been interpreted many ways, and one of the most common is to assume processes, documentation, and planning are not only not valued, they are unnecessary. This practice has the potential to instigate the development of a poorly planned architecture because the architects are never given an opportunity to abstract themselves to a point that they can consider the project goals in their totality. This is due to the fact that the day-to-day effort of preparing to deliver a working product at the conclusion of each sprint is the sole focus of the development team.

Ken Schwaber recognized this flaw and an opportunity to combat it, as he stated in an interview, "Scrum purposefully has many gaps, holes, and bare spots where you are required to use best practices…Scrum then shows you how well that approach works through transparency, so you can continually optimize the approach" (Hansenne and Hibner 2011).

Another concern many have with Agile development is the life cycle's inability to view the "big picture" of the project. Critics suggest that Agile is a license to get busy coding the solution and "learning as you go" as the project evolves, without considering the project from a holistic perspective and hence missing key elements of the architecture requirements. This prospective event would be tremendously detrimental to a system architecture function. Proponents of Agile argue that developers in the waterfall

development get trapped in "analysis paralyses," making determinations early in the project without enough information, on topics that will inevitably change in the future as the project evolves, making virtually any conclusions made superfluous. The dispute is summed up best by Philippe Kruchten, in his article "Software Architecture and Agile Software Development—A Clash of Two Cultures?" when he states:

> They [Agile practitioners] see a low value of architectural design, assert that a metaphor should suffice in most cases and the architecture should emerge gradually sprint after sprint, as a result of a succession of small refactoring. Conversely, in places where architectural practices are well developed, Agile practices are often seen as amateurish, unproven, limited to very small web-based socio-technical systems. (Krutchen 2010, 497)

2. Agile Team Responsibilities

So, who on the scrum team is responsible for the definition and development of the system architecture? This is an important question that must be answered by the team itself.

The scrum team is made up of three roles, each with a distinct set of responsibilities. Teams are self-organizing and intended to be made of multi-disciplinary individuals. Consequently, some overlap of the responsibilities exists. Scrum team roles include the following:

- Product Owner—represents the stakeholders of the project
- Scrum Master—responsible for ensuring that the execution of the Scrum process is executed effectively
- Development Team—responsible for building the system

When contrasted with a traditional system in which roles exist for a system architect and other processes such as quality assurance specialist and configuration manager, it is important to note here that there is no inherent responsibility to any process by any member. According to the Scott Ambler and Associates consulting firm, the embodiment of the function of the system architect in Agile Scum depends on the size of the team (n.d.-a). For a small team, they suggest the role be held collectively by the team (Scott Ambler and Associates n.d.-a). This group suggests that the system architecture function is so important that it should not be held by a single person. Rather, if the team

contributes to the architecture definition, it has multiple benefits. First, it increases the team's knowledge of the system which may be important as development evolves. Second, it may increase the team's acceptance of the architecture since it is a team responsibility. Third, it allows the adaptation aspect of the empirical system to flourish as the team will be more willing to change the architecture if necessary. The authors focus on this point, writing, "When an architecture is developed by the entire team then people are often far more willing to rethink their approach because it's a team issue and not a personal issue" (Scott Ambler and Associates n.d.-a). As the team grows, is not geographically co-located, and/or the team cannot reach consensus, identifying an architecture owner role may become necessary. This person, similar to the product owner, represents the interests of the architecture. However, the authors warn, that this role is not the same as a system architect on a traditional development project. This individual does not work up the architecture plans independently and report back to the team upon completion. Rather, while maintaining the ultimate decision authority, the architecture owner works in partnership with the team to reach decisions. In addition, this person must also continue to accept changes and adjustments to the architecture, allowing adaptation to thrive. He/she will continue to be a functioning member of the team, attend ceremony events (daily scrums and milestone events) and accept opportunities—called Spikes in Scrum vernacular—to investigate new opportunities to make the architecture better. Ultimately, this person will embody all aspects of the empirical Agile Scrum model- visibility, transparency, and adaptation. Adaptation has already been discussed, but the system architect function must also provide visibility into his/her processes and transparency into plans and findings to ensure they remain an effective team member in the overall scrum team.

3. Process Methods

As discussed throughout this thesis, the Agile Scrum process moves very quickly and normally within only a 30-day development window. This creates potential difficulty for the traditional system architect position, as a great deal of time is typically allotted at the beginning of a waterfall development cycle for evaluation of requirements and design prior to developers beginning implementation. To support the initial discovery of

requirements, Agile Scrum prescribes the use of a sprint 0 in which the system architect, heretofore singularly referred to but reflecting an individual or team approach, will take the opportunity to work with the product owner and/or the varied stakeholders and the development team to begin understanding the vision for the product. With this knowledge, the system architect can begin designing an initial plan to support the requirements, build a prototype either physically or virtually, and present it to the product owner and/or stakeholders for feedback. After incorporating this feedback into the initial design or model and ensuring the concept is representative of the goals, the system architect will create a backlog of stories representing the requirements required to complete the architecture. They will then be asked to participate in the planning meetings and development cycle as just another component of the development team. This is not to suggest architectural backlog items require autonomy, rather they should weave feature driven stories within the same sprint (Kruchten 2010, 497). Figure 11 provides a flow chart representation of this process.

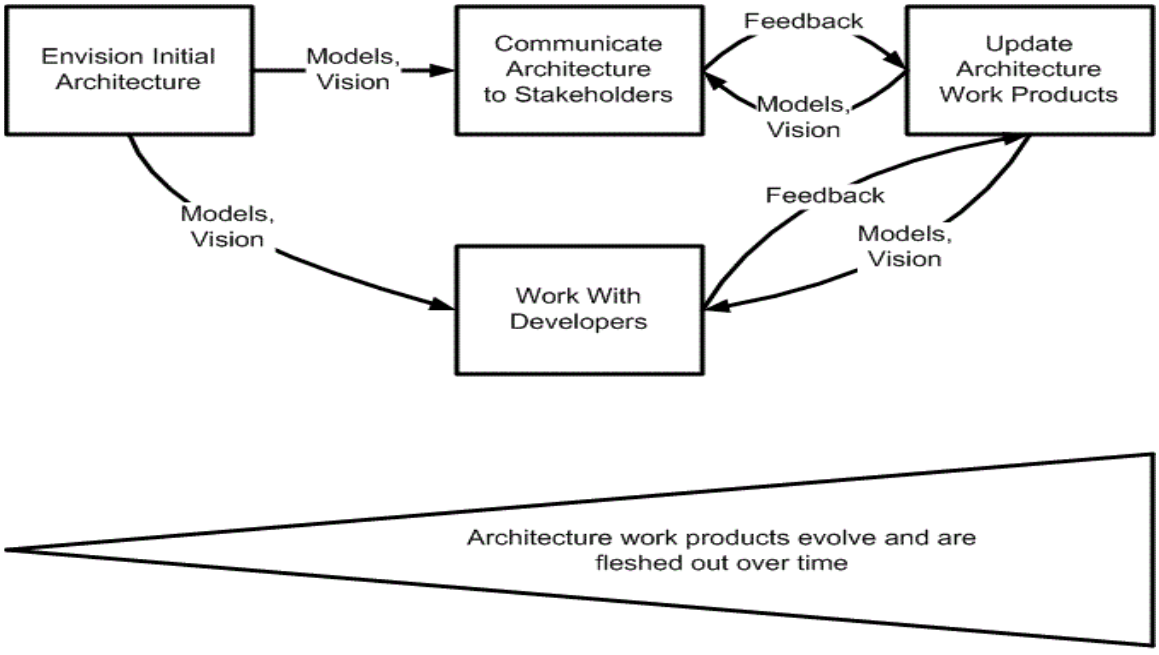

Figure 11. Agile System Architecture Process (from Scott Ambler and Associates n.d.-a)

In addition, the system architect will create his own “Definition of Done” for the work. The “Definition of Done” is simply a listing of required procedures to be completed before a user story is considered concluded from a Scrum perspective (Cohn 2013). This listing should include those items that add value to a product and should be determined before the project begins, collectively by the development team, product owner, and scrum master. Typical elements within a “Definition of Done” may include completed code and unit testing, documentation updates finalized, peer review performed, code checked into the configuration management system, build updated and compiled with no errors, other testing accomplished (e.g., security testing, integration testing, system testing), and Scrum board updated. The system architect’s “Definition of Done” will contain those items that he/she/the team is responsible for developing and/or maintaining. This may include similar responsibilities to the development team’s requirements under the “Definition of Done,” such as updating documentation, ensuring testing is completed and quality assurance requirements are met, performing configuration management, etc.

To ensure holistic views are taken into consideration on an Agile development project, practitioners suggest executing early project in-depth exploration much like the waterfall model. This allows for discovery early in the project, but can be time-boxed within a user story or a sprint to ensure the risk of getting bogged down in analysis is limited. In addition, starting initial user stories at an abstracted level such as “themes” or “Epics” may also help the team achieve a broader view. User stories are written from the perspective of the eventual user of the system in the form of “As a <type of user>, I want <some goal> so that <some reason>.” These stories are meant to capture an individual user action along with the goal and the rationale of the effort. An Epic is simply a large user story, while a Theme is a title given to a backlog item that will decompose into a collection of user stories (Cohn 2012). Using these latter two tools the development team, including the system architect, will ensure a more comprehensive perspective and avoid, or at least address, this pitfall.

Another method to assist projects in ensuring focus remains on the broader picture of the project is using the backlog to the advantage of the development team. The

prioritization of the backlog is a product owner responsibility. However, the development team should be consulted for technical issues potentially affecting that prioritization. This is an opportunity for high risk backlog items that may affect the project as a whole to be worked and resolved. This practice allows possible changes to the system architecture to be raised at the beginning of the project rather than later when additional rework would be required.

The Scaled Agile Framework, a knowledge base made up of contributors from around the world delivering best practices for implementing Agile practices at the enterprise level, provides seven varied principles that should be applied to Agile Architecture (Yakyma and Leffingwell, 2013). Many of these principles have been covered throughout this thesis, so what follows is a synopsis of the additional philosophies. One important principle is ensuring the most difficult aspects of the development are attacked early in the development cycle. Taking care to attack these items early using structured models generated using frameworks such as Department of Defense Architecture Framework (DoDAF), the Open Group Architectural Framework (TOGAF), Zachman Framework for Enterprise Architectures, ensures designs are thoughtful and timely. These methods can be executed in an Agile fashion, incorporating as much ceremony, process, and documentation as is required for the job, if the entirety of the structured process is deemed unnecessary.

Another best practice is modeling and prototyping. Like other development models, using Unified Modeling Language (UML), rapid prototyping, use case diagramming, provides the systems architect with enough information to understand and validate a concept or direction and can provide a suitable Architectural Description Language (ADL). The key element is performing this analysis within time-boxed timeframes (e.g., spikes or sprints) and generating “just enough” documentation to prove out the notion, without generating additional unnecessary overhead (Yakyma and Leffingwell, 2013). At times, photographs of whiteboards are taken as the documentation and placed in planning documents, while at others, more formal products are prepared. Figure 12 portrays some examples of these artifacts.

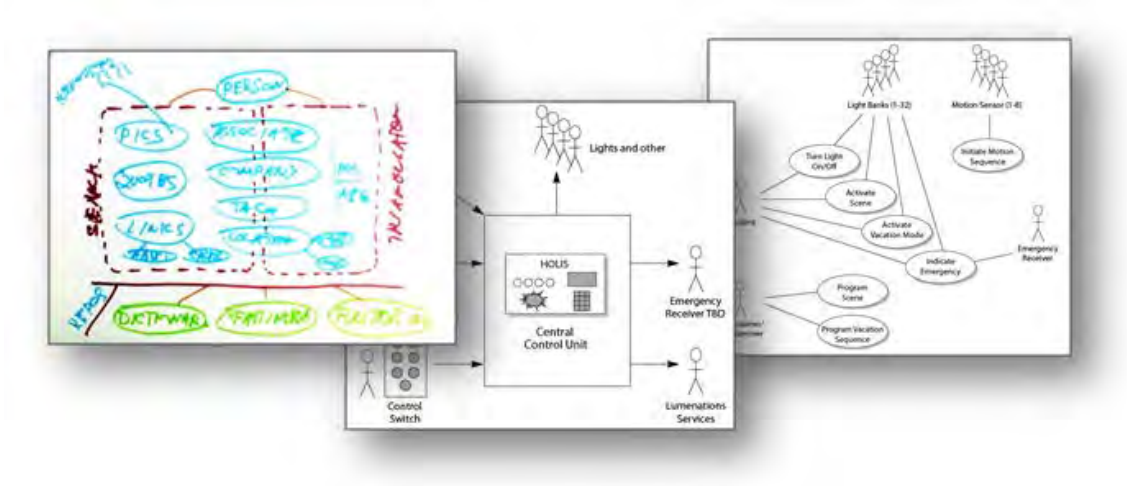

Figure 12. Examples of Artifacts for an Agile Development (from Yakyma and Leffingwell, 2013)

C. CHAPTER SUMMARY

This chapter explored the Agile implementation of applying an architecture to a system. Various viewpoints exist across the community showing both the implication that the Agile method overlooks or skips this discipline, and the prospect of using the Agile ceremonies to assist in generating a robust architecture implementation. One of the big differences demonstrated in this chapter was how the Agile approach differs from the traditional manner in that the system architecture is revisited with every sprint iteration, rather than only at one point in the project. However, critics argue the project is never considered holistically generating its own set of issues (Krutchen 2010, 497). Agile role definitions may also assist, such as the team collectively performing the role of the architect, or assigning a small team to the task. There are many processes for implementing the architecture function, including incorporating a sprint 0 into the Scrum model, including the architecture element into the "Definition of Done," and helping the team gain a broader view of the project by abstracting the requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RISK MANAGEMENT

A. INTRODUCTION

There are many opportunities to apply risk management processes to the Agile Scrum development model. However, even without any enhancements, the process drives down risk inherently. Figure 13 graphically portrays how the Agile Scrum process takes on risk early in the project.

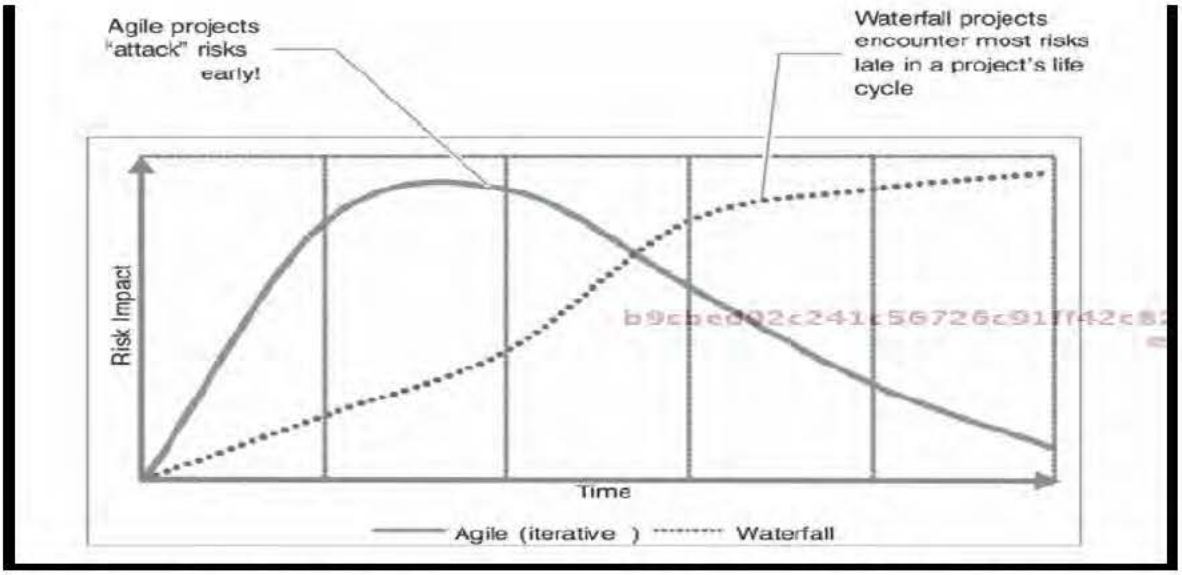

Figure 13. Agile Inherently Drives Down Risk (from Whitaker 2010, 254)

Agile drives down risk in a number of ways. Some of these reasons are outlined below.

- The Agile Scrum process forces a prioritized backlog, which allows for the riskiest elements of the projects to be confronted at the appropriate interval. Many times that may mean very early in the development process, while at others it may mean after a development milestone has been reached. This is an opportunity for high risk backlog items that may affect the project as a whole to be worked and resolved. This practice allows possible changes to the system to be addressed at the beginning of the project rather than later when additional rework would be required. Figure 14 illustrates how probability weighting can be applied within an Agile process. If one considers the blocks as backlog items, the backlog can be dissected as a path through the project and any route can be taken

to drive down risk; as opposed to other processes in which the plan is more of avoiding obstacles than actively selecting the way ahead.

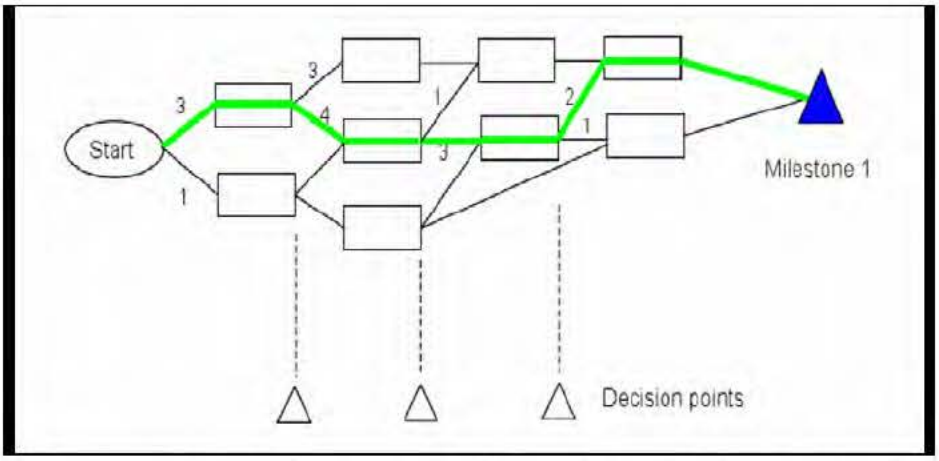

Figure 14. Example of Agile Selecting Path through Project (from Chin 2004, 128)

- Stakeholders often do not always know exactly what they want. This being the case, many projects run the risk of building the wrong solution. The Agile Scrum process allows the flexibility to change direction on a sprint-by-sprint basis. If the Stakeholders arrive at the sprint review meeting to see the deliverable and inform the group that this is not the direction the product should be heading, or likewise informs the team of a new direction due to a change in the operational landscape, Agile Scrum is built to accept this adjustment and continue the development process with little impact.
- Development teams do not have to understand every requirement perfectly before beginning work. One of the big differences between Agile and other models is that full scale development begins early. While this offers some potential drawbacks discussed later in this thesis, it also offers the team the ability to confront the understood risk elements immediately.
- In terms of schedule, planning in small increments allows for greater opportunities to meet schedule requirements. While all projects can plan for years in advance, the probability of incorrectly identifying a schedule grows with the length of time covered by the plan. Simply, a plan for a year from now is not as good as a plan for six months from now. This six month plan, in turn, will not be as accurate as a plan for three months from now, which will not be as accurate as a one month plan compared to a two week plan, or a plan for just one day. Therefore, Agile Scrum does not spend a lot of time on far reaching plans because of the known inaccuracies. Rather, the model plans for general accomplishments in the

future and gets more granular as the timeframe becomes more and more narrow.

- Projects are tested early and often. Unlike many other models that perform unit testing early and potentially some integration testing as the product is developed, Agile Scum incorporates full scale testing with every delivery which occurs at the conclusion of every sprint. This testing forces issues to be raised early in the process and handled immediately, rather than waiting for later in the project when risk is more difficult to resolve.
- Daily Standups are Agile Scrum meetings that are held every day with a strict time limit of 15 minutes. The members of the team share what they accomplished yesterday, what they intend to accomplish today, and any impediments to their plans. This communication permits issues encountered by the development team to be resolved on a daily basis rather than waiting for the weekly, bi-weekly, or monthly meeting, or simply waiting for the developer to independently raise their concerns. In addition, these Daily Standups allow increased communication with the Product Owner, who is acting as the Stakeholder representative. This exchange allows for the Product Owner to hear the daily evolution of the development and provide answers to questions and/or assistance in removing impediments as required.
- The conclusion of any sprint offers a shippable product as its artifact. This being the case, users are able to see the development of the project and provide feedback early and often in the development cycle. Again, this early feedback and communication allows for the right product to be developed as feedback can be incorporated immediately without the huge impact that confronts other methodologies which wait much later to provide a product for feedback. Jamie Cook, author of *Agile Productivity Unleashed*, substantiates this notion as she writes, "Agile approaches mitigate the risk of situations like these [creating mock-ups for examples] occurring by requiring the delivery team to do the *actual work* required before presenting outputs to business owners. Every interim deliverable represents a slice of the final deliverable, including all of the work required to make it a production-ready output. Knowing the hurdles upfront means that the risk of having insufficient information, resources, time or finances is substantially mitigated" (Cooke 2010, 190).
- Retrospectives are held at the end of each sprint cycle. This short ceremony is a simple process including all of the members of the Scrum team to share what worked well in the sprint and what needs improvement. This is then documented and the changes discussed are applied immediately to the next sprint. Again, this early identification of issues not only makes for a better development environment, but also drives down risk by improving the process.

Another element Agile brings with it is empowering the scrum team. The transparent nature of the Agile effort allows stakeholders and business executives alike the opportunity to view what is going on at any point in time to assess the business value the team is producing. One source suggests:

> This accountability is not due to the fact that the day-to-day tracking of Agile work provides a 'big brother' opportunity for executives to track every detail of their employees' work. In fact, Agile approaches can produce the exact opposite effect, by instilling an unprecedented level of trust in the work of their employees. (Cooke 2010, 275)

The author goes on to state that with this level of transparency, key decisions can be made from more precise information allowing the organization to be "…protected from the risk of executives making decisions based on faulty (or misleading) information" (Cooke 2010, 275). In addition, Agile Scrum incorporates the development team into the planning and time estimation process which assists in the team committing to the given schedule. This confidence provided to the development team from the executive level generates two opportunities. First, it enforces the notion of self-management for the team. This simple concept allows teams to thrive. Second, the positive impact reminds the team that this faith provided by the executive level can be revoked, so overestimating or underestimating and not proving enough return on investment has the potential to cause this trust to be revoked. The concept leads an author to state:

> This [confidence from management] creates an imperative for Agile teams to remain vigilant in their ability to accurately report and deliver on business value to the organization. The fact that the process is self-correcting can give senior management the confidence of knowing that risk is being actively managed at all levels of the organization. (Cooke 2010, 276)

Along these same lines, another author asserts that the development team should be given the leverage to adapt the Agile model when applying risk management principles, "Train and empower project teams to make good risk management decisions to tailor the life-cycle model as needed to develop an appropriate balance of control and flexibility for each project" (Cobb 2011, 92).

B. APPLYING RISK MANAGEMENT TO AGILE SCRUM

This section reviews the various challenges and opportunities encountered when applying risk management practices when using Agile Scrum. It defines the founders' view of this practice along with varied industry views. Next, it defines the responsibilities of the team members to implement this practice, and provides descriptions of both qualitative and quantitative methods that may be exercised.

1. Challenges and Opportunities

While the positive elements of applying the Agile model to risk management concepts abound, there are cons to this approach as well. The author/founder of the Agile Scrum model, Ken Schwaber, recognized the intentional shortcomings of the Scrum approach and the opportunity for optimization in this quote. "Scrum purposefully has many gaps, holes, and bare spots where you are required to use best practices—such as risk management. Scrum then shows you how well that approach works through transparency, so you can continually optimize the approach." (Hansenne and Hibner 2011). The negative identified in this quote is recognizable in the way many Agile projects are managed. While there are best practices Mr. Schwaber is recognizing here to fill in the blanks in the model, in many Agile implementations this step is ignored. For example, Agile is intended to be a lightweight documentation and process model. Agile factions have gone as far as to coin a new acronym to define these issues with other development methods, pointing to the analysis and documentation that accompanies it, stating it is generally YAGNI. Consequently, many implementations take this as license to ignore essential processes. So, a risk management plan, for example, is not always a priority and believed to be required. Agile is also designed to be executed as a rapid development process. This being the case, risks are often ignored or overlooked. In an Agile environment, identification of risk must happen early as the time horizon of mitigation and contingency approaches become an issue if risk is only evaluated around sprint cycles. As described previously, Agile Scrum inherently incorporates a backlog-centric process in which risk can be given proper visibility and evaluated accordingly. However, if the management of the process does not assess risk when prioritizing the

backlog, this opportunity is lost and risk is effectively ignored. Also, in Agile Scrum, acceptance criteria are used as the basis on which backlog items are evaluated for completion. Acceptance criteria are authored by the stakeholders, or product owner on their behalf, and define what they are looking for in the capability the backlog item represents before the item will be declared complete. If taken at face value, acceptance criteria are simply a list of tests to evaluate the capability, but used effectively as a best practice to identify risk management concerns, this becomes another tool to provide focus on the risk factors of the project.

Simply stated, for many of the issues identified above, executing the Agile methodology intrinsically creates a best practice risk management strategy. However, more attention is required. Enforcing focus on the risk management plan can provide fundamental project risk focus for the Agile scrum team. This enforcement must be generated from not just within the scrum team, but also from the stakeholders. Merely establishing importance of risk management concepts will help in staving off the tendency to ignore this planning phase. In addition, employing appropriate attention to risk management during the development phases by distinguishing times within the sprint when risks will be evaluated will ensure this practice is not overlooked. Review of risk may happen at the beginning, during the sprint planning session, during daily standups, or at a designated time during the sprint cycle. Regardless, placing importance on the topic will assist in ensuring it receives a higher degree of scrutiny. Using the backlog to the advantage of the team's risk management focus is the job of all members of the team. Stakeholders must apply importance to this task from a business perspective, and the team must follow suit, identifying technical risk areas that may adjust prioritization. Finally, incorporating risk components into acceptance criteria can become a best practice if the team chooses take advantage. Again, with the stakeholder recognition of risk items in the authoring of the initial criteria, and the development team responding with a risk slant during the evaluation of the acceptance criteria, this can be a powerful opportunity to address risk.

Other opportunities for applying risk management notions in Agile are:

- Incorporating the self-organizing team concept and instituting peer review and pair programming as a best practice. Each of these tools allow for both the growth of cross functional teams, and a more defect free implementation. Both outcomes are positive from a risk perspective. Creating team members that can step into a variety of positions reduces the risk of employee turnover impacts, while reduction of defects reduces the probability of system failure.
- Management is more accepting of an introduction of change. As stated previously, Agile is built to allow incorporation of change to the product baseline at any point during the development process. Management understands this concept and is much more alacritous to accepting a direction change, than with other models in which either the injection of a new requirement or direction will introduce enormous cost or schedule implications. According to one author, "To be successful, we need to break away from the stigma that change is bad, or is the result of bad planning. Creating, communicating, and discussing appropriate risk management plans is an effective way to do this" (Chin 2004, 131).
- Transparency of the development environment and schedule through information radiators, both ad hoc and formal, offers more information earlier in the development process to stakeholders and executives for decisions to be made upon. In addition, quick and repetitive product deliveries provide tangible output to determine both if the right product is being developed and if the investment in the development is returning value. Overall, this allows the project an opportunity to fail early, saving the company both investment assets and time.

2. Agile Team Responsibilities

So, who on the scrum team is responsible for risk management? This is an important question that must be answered by the team itself.

The scrum team is made up of three roles, each with a distinct set of responsibilities. Teams are self-organizing and intended to be made of multi-disciplinary individuals. Consequently, some overlap of the responsibilities exists. scrum team roles include the following:

- Product Owner—represents the Stakeholders of the project
- Scrum Master—responsible for ensuring the Scrum process is executed effectively
- Development Team—responsible for building the system

When contrasted with a traditional system in which a role potentially exists for an individual or team performing risk management functions (i.e., risk management specialist, and other roles such as quality assurance specialist and configuration manager), it is important to note here that there is no inherent responsibility to any process by any member.

Accordingly, from a risk management perspective, while the Product Owner represents the stakeholders, the role is to also ensure risk management is a priority on the project. This includes placing emphasis on the generation of a risk management plan, and incorporation of risk in both the acceptance criteria and the "Definition of Done." The concept of acceptance criteria's role in risk management has already been reviewed, so this opportunity will be taken to review the role of the "Definition of Done." As described in Chapter III, Requirements Management, the "Definition of Done" is simply a listing of required processes to be completed before a backlog item is considered completed from an Agile Scrum perspective (Cohn 2013). While placing importance on this concept drives down risk in and of itself, over-arching risk functions may be placed in this definition as well. Items such as updating qualitative methods such as a Risk Task Board or identifying criteria for quantitative methods may be defined here. These processes will be discussed in the section below.

The scrum master and development team have corresponding tasks in the risk management practice on the scrum team. The scrum master ensures the risk management process is executed and remains highly visible, not just within a given sprint but throughout the project life cycle. This individual is also responsible for training and empowering the development team to make good risk decisions, and holds the development team responsible for planning and execution of the tasks. The development team congruently responds with thoughtful and valid estimating of tasks and honestly reporting on their own work efforts. In addition, the development team must adhere to both the defined acceptance criteria and "Definition of Done."

Overall, the entire Agile scrum team is responsible for application of the risk management process on the project, but from varied perspectives. While the product owner is executing the task from a business and stakeholder perspective, the scrum

master is performing from a process and execution angle, while the development team executes from a development and technical viewpoint, as depicted in Figure 15.

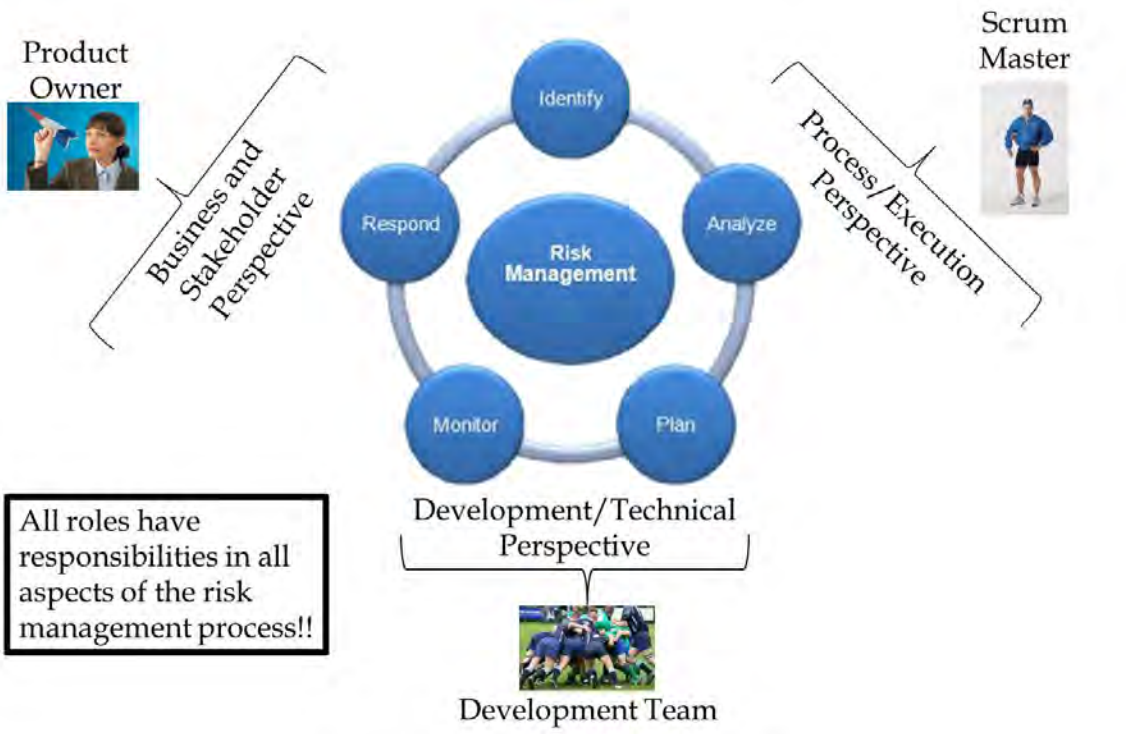

Figure 15. Scrum Roles and Risk Management (Risk Management Life cycle image from Aravinda 2010)

3. Process Methods

When applying risk management approaches to project management, there are both qualitative and quantitative techniques that may be practiced. This section provides descriptions of implementation options for each.

a. Qualitative Methods

Qualitative risk methods are subjective in nature, using subject matter experts and others to provide opinions on possible risk areas and outcomes. In the Agile methodology, any qualitative method can be applied as necessary to execute the project's risk management approach. In this section, both Agile–based methods and more traditional methods with Agile twists will be described.

First, the Agile methodology will be explored. Examined several times throughout this thesis are the information radiators that dominate the Agile war room. These ad hoc data centers are built by the scrum team and include material such as sprint status, burndown charts tracking project progress in graph form with time represented along the X axis and tasks on the Y axis, task boards, team rules, retrospective results, and the like. Another type of information radiator that can be created is the Risk Task Board. This concept is presented in an article in *Software Development* magazine by Preston Smith and Roman Pichler in 2005. In this writing, the authors suggest the following process:

1. Review the product backlog and perform risk identification. In this exercise all members of the Agile scrum team scrutinize the backlog and assist in identifying risks. As these are found and documented on 3x5 cards or sticky notes, they are placed on the board.
2. Analyze and prioritize the risks. Each team member then offers their rationale for the risks posted on the board.
3. Risks are then categorized and prioritized by team vote. Using participatory decision making, the risks are individually accepted or rejected, with those that are accepted prioritized accordingly. This concept harkens back to earlier discussions regarding the empowered team and individuals owning the risk if they identify them.
4. Risks are mapped back to the product backlog. Some identified risks are then converted to backlog items themselves, while others are applied directly to affected backlog items.
5. Product backlog is then reprioritized to determine release and sprint goals. Once the risk backlog items are all within the backlog, the entire backlog is reprioritized taking these risk areas into consideration.

Figure 16 provides a picture of the process.

Figure 16. Creation of the Risk Task Board (from Smith and Pichler 2005, 53)

A more traditional qualitative method that works well in the Agile environment is the preparation of a Risk Management Plan with the incorporation of a Risk Breakdown Structure (RBS). Ken Whitaker (2009), in his book *Principles of Software Development Leadership: Applying Project Management Principles to Agile Software Development* provides a detailed description of the process using these two elements and combining them for the benefit of the Agile project. Mr. Whitaker reviews the contents of the Risk Management Plan suggesting it should contain general data, such as process definition, role identification, and timing and ceremonies for tracking risks. He then suggests creating an RBS based on categorization. He states that in this diagram, the project is not trying to resolve the risks, "…instead, you are categorizing the risks that could trip up (or accelerate) your project. As a result, reviewing the RBS diagram should become an agenda item for every team planning meeting" (Whitaker 2009, 114). Figure 17 is an example of an RBS.

Figure 17. Risk Breakdown Structure (from Whitaker 2009, 115)

Next, the author recommends starting the process of documentation by identifying the risk. This is accomplished by applying a risk number and a description to each risk as depicted in Figure 18. Like the Agile process itself, this is an iterative activity that should be revisited as the sprint cycles evolve. In addition, like the creation of the risk task board described above, this action should be accomplished with full team participation. Again, this creates buy-in by the development team and allows them to take ownership of each risk. Various tools may be used by the team to help reach a collective consensus using techniques such as brainstorming, Delphi, and strengths, weaknesses, opportunities, threats (SWOT) analyses.

Category	Risk Topic	#	Risk
Technical	Requirements	1	Database management system may not support automated security labels.
External	Regulatory	2	Database management system may not pass certification.

Figure 18. Identifying and Documenting the Risks (after Whitaker 2009, 121)

Upon identification, the risks are entered into the risk register and mapped to the RBS. The risk register is then updated with qualitative methods and any other risk processes as shown in Figure 19.

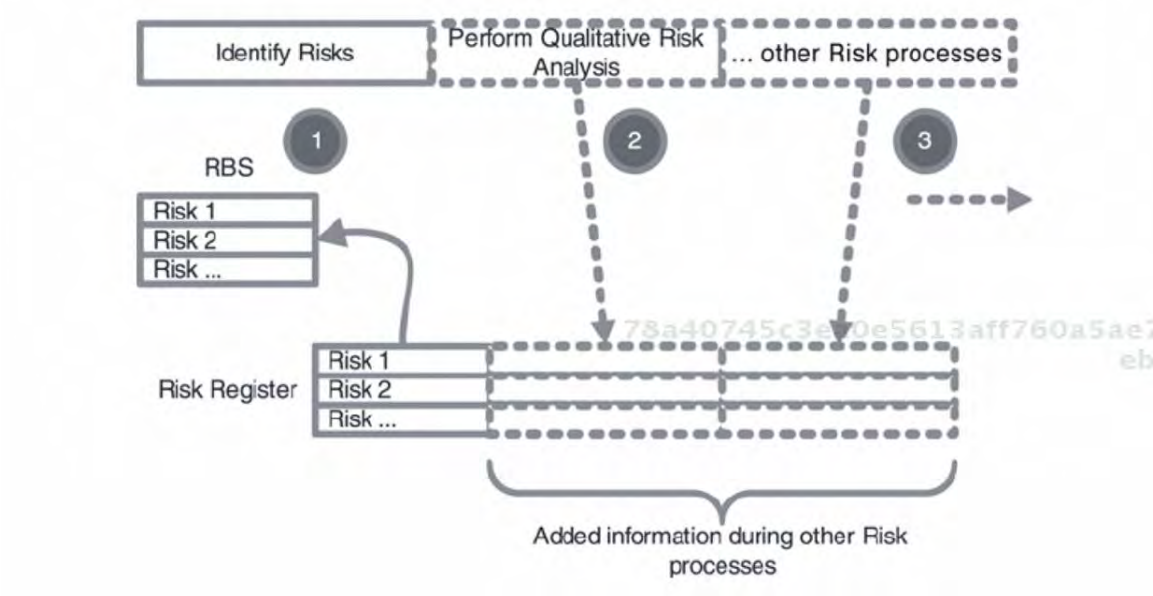

Figure 19. Risk Register Mapped to RBS and Updated (from Whitaker 2009, 122)

Next, the risks are analyzed and annotated with additional attributes describing each line item. These include the following data points:

- T/O—Threat/Opportunity. This categorizes each risk as either a threat to a project, or an opportunity which could ultimately benefit the project over time.
- P—Probability. This is defined as a subjective High (H), Medium (M), or Low (L) and is based on parameters established by the project.
- I—Impact. This is also an H, M, and L value and assesses the risks impact if realized.
- U—Urgency. Also, an H, M, and L value and relates to how crucial the risk is to the project in terms of timing or other potential factors.
- Priority—An H, M, and L value that establishes the priority of the risk in relation to the other identified risks.

Figure 20 is an example of a risk register entry with these values identified.

Category	Risk Topic	#	Risk	T/O	P	I	U	Priority
Technical	Requirements	1	Database management system may not support automated security labels.	T	M	H	H	H
External	Regulatory	2	Database management system may not pass certification.	O	H	M	H	H

Figure 20. Risk Register with Attributes (after Whitaker 2009, 124)

From this risk analysis, quantitative methods and decision analysis can be applied. This will be discussed in the next section.

b. Quantitative Methods

Quantitative risk management methods are also used in Agile. Several examples will be reviewed below, again those applicable to any methodology but adapted for Agile and those closely fitted to the Agile framework.

Using the previously mentioned risk register example, Ken Whitaker next applies a quantitative methodology. Three new values are introduced to the model. These are as follows and illustrated in Figure 21:

- P%—Probability of threat/opportunity occurring. This applies a percentage value based on various analysis techniques such as sensibility and Monte Carlo.
- I$—Impact (relative cost) of the threat/opportunity occurring. This is the projected cost the project will incur if the risk is realized.
- EMV—Expected monetary value analysis = (P%)(I$) [- threat / + opportunity]. This value is the probability percentage multiplied by the cost impact, and then presented as -/+ depending on whether or not the risk is a threat or an opportunity.

Category/ Risk Topic	#	Risk	T/O	P	I	U	Pr	P%	I$	EMV
…	1	Database management system may not support automated security labels.	T	M	H	H	H	60%	$150,000	-$90,000
…	2	Database management system may not pass certification.	O	H	M	H	H	30%	$200,000	$60,000

Figure 21. Applying Quantitative Value to Risk Register (after Whitaker 2009, 127)

With all of these EMVs calculated, the project can develop an EMV for the project as a whole. These can be fed back into the projected total project cost. These values can then be applied to decision trees. For example, suppose the project decided to choose between the threat and opportunity risks:

- Database management system may not support automated security labels.
- Database management system may not pass certification.

In order to do this, the project would first have to pose a question on which to base a decision. In this case, what would be the financial impact of attempting to execute the waiver process for the database management system? In the decision tree both the positive limb is defined (yes, attempt the waiver process) and the negative is defined as well (no, integrate the new database into the project). From this, a node is reached in which a cost is defined. On the positive side of the tree there is a cost of $100,000 to pursue this path. However, on the negative side, there is an associated cost of $200,000. The probability of success on attempting the waiver process is 50/50, while integrating a new database is 90/10. Each one of these probabilities is also associated with an equation, EMV x Probability. Thus, the following equations are defined:

- Successfully achieving the waiver process—$60,000 x 50%
- Unsuccessfully achieving the waiver process—$60,000 x 50%
- Successfully integrating a new database—$60,000 x 90%
- Unsuccessfully integrating a new database—$60,000 x 10%

Figure 22 depicts the decision tree described above.

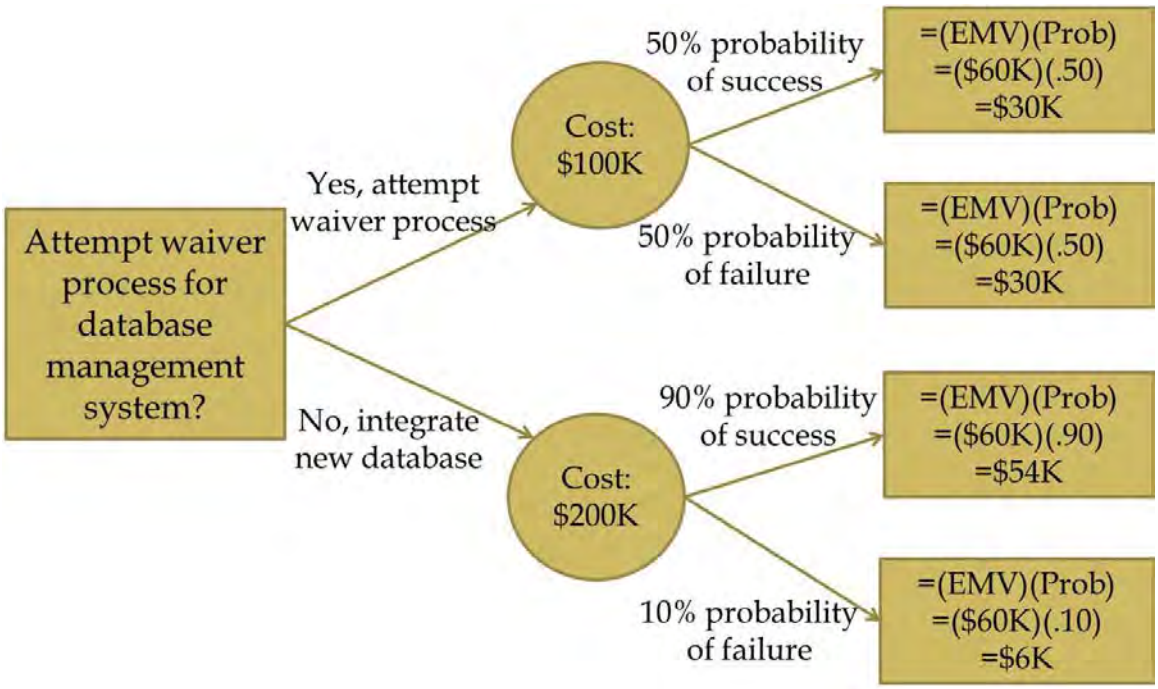

Figure 22. Decision Tree Example (after Whitaker 2009, 129)

These values can then be placed into a calculation depicting both success and failure of the two paths. Figure 23 depicts the success calculation. Successfully executing the waiver process for the database management system, and integrating a new database both have sunk investment costs, shown as negative values below, of $100K and $200K respectively. Each has an EMV; $30Kif the waiver is successful and $54Kif database integration is successful. These values are positive, as depicted below, and are used to reach a summation when added to the cost value. Each generates a total cost value and a recommended way ahead.

Success Calculation			
Attempt waiver process for database management system?	Cost	EMV	Total Cost
Attempt waiver process	-$100K	$30K	-$70K
Integrate new database	-$200K	$54K	-$144K
Attempting waiver process is less costly by			$74K

Figure 23. Decision Tree Success Branches (after Whitaker 2009, 131)

The next illustration, Figure 24, depicts the failure calculation. Failing to achieve the waiver for the database management system and failing to integrate a new database are the two reciprocal scenarios. Again, these both have the same sunk investment costs, shown as negative values below, of $100K and $200K respectively. Each also has an EMV; $30K if the waiver is unsuccessful and $6K if database integration is unsuccessful. These values are positive, as depicted below, and are then used to reach a summation when added to the cost value. Each generates a total cost value and a recommended way ahead.

Failure Calculation			
Attempt waiver process for database management system?	Cost	EMV	Total Cost
Attempt waiver process	-$100K	$30K	-$70K
Integrate new database	-$200K	$6K	-$194K
Attempting waiver process is less costly by			$124K

Figure 24. Decision Tree Failure Branches (after Whitaker 2009, 131)

Armed with this information, a decision maker can now use this data to determine the best way ahead for this particular issue.

Another method that can be applied to assist in prioritizing the backlog in the Agile method is adding a detection adjustment. According to Gary Chin, in his book *Agile Project Management: How to Succeed in the Face of Changing Project Requirements*, "Since the basic R [Risk] = I [Impact] x P [Probability] model can't possibly capture all of the nuances of a particular project situation, it is common to add a qualitative adjustment factor to the risk ordering" (Chin 2004, 135–136). Detection is the term the author uses to label this adjustment. It simply identifies the ability of the risk to be detected: -1: Easy, 0: Hard, 1: Impossible. This additional value can create separation in the backlog when prioritization is executed and can be defined as: Risk = Impact x Probability + Detection Adjustment. Mr. Chin goes on to suggest that the goal of this exercise is to break ties between competing risks. He states that if any additional

qualitative adjustments are available, those should be added to the equation as well, as in: Risk = Impact x Probability + Detection Adjustment + Qualitative Adjustment.

An additional method geared toward iterative development models such as Agile is the EVOLVE+ prioritization method. This method was conceived by two individuals, one from the University of Calgary (Canada) and the other from Queens University Belfast (United Kingdom) who took on the task of mathematically determining the right requirements to be executed in the right order within the right incremental release, while still satisfying all participating stakeholders and reducing overall risk to the project. The culmination of this study was the EVOLVE+ method. This method identifies a number of variables and feeds them through a set of complex equations. The resulting values categorize which requirements fit best into which increments, in which order, while driving down risk to an acceptable level, and returning the highest benefit (Ruhe and Greer 2003). Figure 25 displays an example of the end result of these equations.

Probability	Benefit (A)	Risk	Effort	Release	Assigned Requirements
95%	201	0.56	23.4	1	r_1 r_2 r_8
		0.82	24.0	2	r_5 r_7 r_{10} r_{14} r_{16}
		1.08	24.4	3	r_3 r_4 r_6 r_{12} r_{15} r_{17}
		0.64	24.7	4	r_5 r_7 r_{10} r_{14} r_{16}
		0.24	13.4	5	r_8 r_{20}
90%	218	0.84	23.8	1	r_1 r_2 r_7 r_{18}
		0.89	24.3	2	r_4 r_5 r_8 r_{14} r_{16}
		1.05	23.9	3	r_3 r_6 r_{11} r_{12} r_{13} r_{15}
		0.45	23.4	4	r_9 r_{10} r_{17} r_{19}
		0.11	9.66	5	r_{20}
80%	224	0.9	24.6	1	r_1 r_3 r_{11} r_{12} r_{13}
		0.91	24.5	2	r_2 r_{14} r_{15} r_{16} r_{17} r_{18}
		1.00	23.6	3	r_4 r_5 r_7 r_8 r_9
		0.44	20.9	4	r_6 r_{10} r_{20}
		0.09	4.7	5	r_{19}

Figure 25. EVOLVE+ Result for Release Planning (from Ruhe and Greer 2003, 8)

This data can then be evaluated to identify, given a success probability of not exceeding the effort bound and benefit scale, which requirements should be executed in what release order. The data can be further evaluated to determine what the effort will be

for the development team and the risk interval that will be assumed. The team can then use the results to select the appropriate requirements to attack within each release to provide the greatest benefit within acceptable risk intervals and probabilities. While this process may take a significant level of effort to set up and execute, it is a rare tool that takes into account so many facets of the incremental process, allowing the team to choose from the most promising solutions that fit within the parameters characterized by the individual project.

C. CHAPTER SUMMARY

Risk management is an integral process component to any software development strategy. This management strategy ensures system development risks are acknowledged, understood, evaluated, and that "…appropriate strategies are undertaken to mitigate and control the risks identified" (Misra et al., 2007, 62). The Agile development methodology inherently maintains cognizance of much of this discipline. Following the Agile Scrum method, prioritizing the backlog placing higher priority (riskiest) capabilities early in the development process, changing project direction each sprint, building in small pieces, and testing early and often throughout the project life cycle, are some of the ways in which this is accomplished. However, more overt processes may also be executed such as reviewing identified risks on a predetermined basis and incorporating risk statements to resolve into the acceptance criteria. Like other practices, all members of the team are responsible for maintaining awareness of project risks, from the product owner working on behalf of the stakeholders from the project domain perspective to the development team using the "Definition of Done" as a guide.

There are both qualitative and quantitative methods that can be applied in the Agile development methodology. From a qualitative perspective, this may include a Risk Board or a Risk Management Plan; while from a quantitative view, this may incorporate applying quantitative measures to a qualitative risk register using a numbered risk adjustment label to quantify or a more robust process such as EVOLVE+.

Using these approaches, not only is risk management important in the Agile paradigm, but can also be improved. As one author writes:

Through a solid understanding of agile principles and values, we can adapt the risk management process to exploit the strengths of the agile approach used. By looking at a project through agile eyes, we can manage the worst 80 percent of its risks with 20 percent of the effort expended in conventional approaches. (Smith and Pichler 2005, 53)

VI. TEST MANAGEMENT AND PROJECT DOCUMENTATION

A. INTRODUCTION

In order to ensure quality assurance in a software product the testing and documentation approach and methods must be sound. This is as true in Agile development as in any other development methodology. However, the Agile Manifesto leads some projects to believe it is unnecessary. The manifesto reads that the signers value "Individuals and interactions over processes and tools" as well as, "working software over comprehensive documentation" (Agile Manifesto 2001). One author and developer of 30 years, notes the documentation side of this writing:

> Like finicky domestic helpers who announce that they 'don't do windows,' I've often heard software developers state proudly that they 'don't do documentation,' even refusing lucrative job offers that stipulated it. This viewpoint is echoed in the widely praised Agile Manifesto. (Selic 2009, 11)

Another author writes, "While some teams do seem to use the 'agile' buzzword to justify simply doing whatever they want, true agile teams are all about repeatable quality as well as efficiency" (Crispin and Gregory 2009, 15). So, from this it can be surmised that while Agile practices can be used as fodder to avoid the dreaded overhead of these disciplines, they can also be used to ensure quality.

B. APPLYING TEST MANAGEMENT AND PROJECT DOCUMENTATION TO AGILE SCRUM

This section tackles the challenges and opportunities encountered when incorporating test methods and project documentation using Agile Scrum. It defines how these practices may be interpreted for use in the traditional and Agile software development life cycles (SDLC), the responsibilities of the team members to apply this practice in a project, and provides process methods by which each may be executed.

1. Challenges and Opportunities

One of the biggest differences between the Agile and waterfall methodologies from a testing perspective is the timeframe in which the testing is executed. As pictorially presented in the Figure 26, in a traditional development model the testing phase is reserved for the end of the process, while in an Agile process is iteratively executed (Crispin and Gregory 2009, 12).

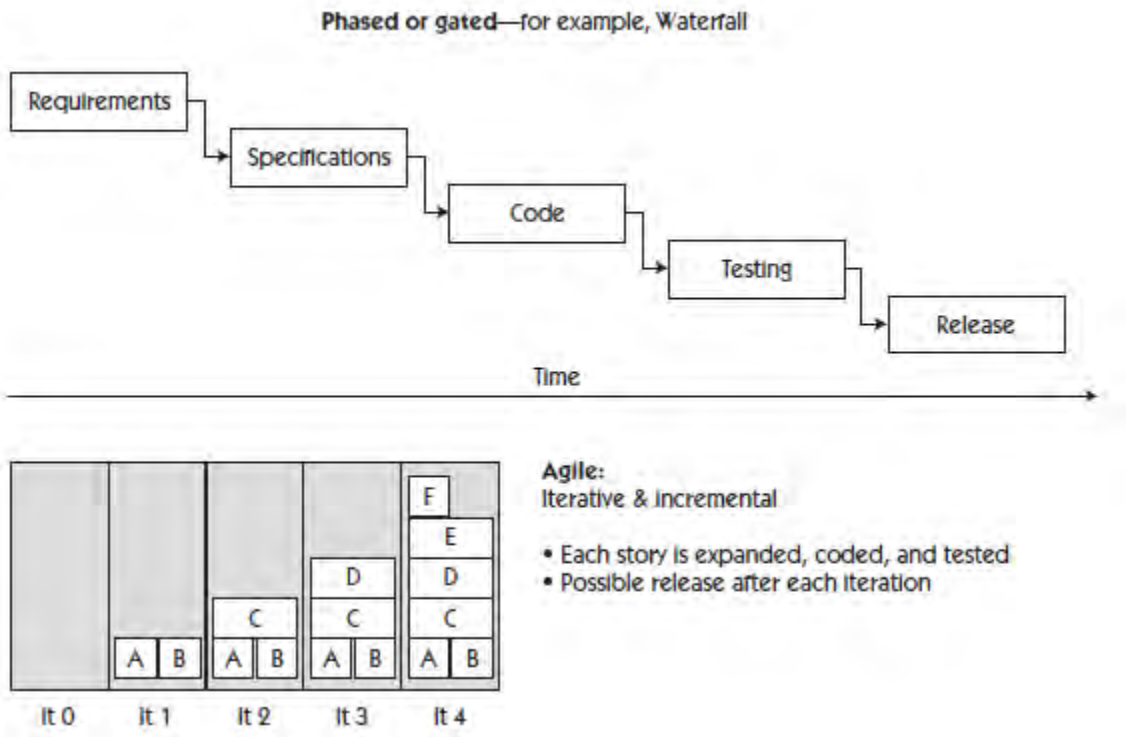

Figure 26. Traditional Testing versus Agile Testing (from Crispin and Gregory 2009, 13)

This element poses a challenge and an opportunity simultaneously depending on the group that implements it. It creates a challenge to testers new to Agile because these individuals were used to being the "…gatekeepers of quality…" and "… had the power to postpone releases or stop them from going forward" (Crispin and Gregory 2009, 9). While the power still exists theoretically, the execution of the tasks are much more of a partnership. The testers and developers work together, ensuring the two keep pace with each other. This is where the opportunity arises. From a requirements perspective, in the

traditional model testing documentation were created from requirements generated early in the development process with months or even longer passing before tests are authored (Crispin and Gregory 2009, 13). While in the Agile process, tests are written "…that illustrate the requirements for each story days or hours before coding begins. This is often a collaborative effort between a business or domain expert and a tester, analyst, or some other development team member" (Crispin and Gregory 2009, 14). Furthering the collaboration concept, these authors go on to explain that, "Testers might pair with other developers to automate and execute test cases as coding on each story proceeds" (Crispin and Gregory 2009, 14). Another set of authors also reference this concept of collaboration in their study of "Agile Software Testing in a Large Scale Project," writing, "In a traditional project, "everyone is responsible for quality," but in Agile projects, everyone actually writes tests. This includes all developers, business analysts, and even the customer" (Talby et al., 2006, 32). This study goes on to state:

> Having everyone test offered several project advantages. First, it eliminated the team's reliance on the single, assigned project tester, who would have otherwise constituted a major bottleneck. This proved highly important because—for reasons unrelated to the project—the tester was replaced twice during the first three releases. Second, because developers were responsible for writing tests for each new feature, their test-awareness increased and they prevented or quickly caught more edge cases while they worked. Finally, because people knew that they were required to test their specifications and code, they designed them for testability. That is, where appropriate, developers added special features or design considerations to the code to make hard-to-test features testable. Such coordination tasks are often difficult and neglected when developers and testers work as separate teams. (Talby et al., 2006, 32)

However, not all agree with this statement. One source believes there is a reason the testing and development should be done separately, writing "A programmer wants his code to work, whereas a QA [quality assurance] tester wants it to die—he actively wants to find bugs, errors, and ugliness. One person just can't wear both hats at once; it creates a major conflict of interest" (Stevens and Rosenberg 2003, 359). These authors do go on to acknowledge that this relationship can be successful, if individual ownership is required on the project forcing unit tests to be required to 100 percent completed as part of a developing process (Stevens and Rosenberg 2003, 359).

Another opportunity the Agile process provides is the process itself. In the words of one author, “The team builds and tests a little bit of code, making sure it works correctly, and then moves on to next piece that needs to be built. Programmers never get ahead of the testers, because a story is not ‘done’ until it has been tested” (Crispin and Gregory 2009, 12). This reference to “done” is the same as the discussion throughout this thesis to the “Definition of Done.” As discussed previously, this concept is a vital component to the Agile process. In this case, it guarantees time is devoted to testing rather than what oftentimes happens in the traditional process, waiting until the end of development for the test phase where time has been stolen throughout the rest of the project, which causes less time to be spent on testing. The authors providing Figure 25 sums this issue up:

> The diagram is idealistic, because it gives the impression there is as much time for testing as there is for coding. In many projects, this is not the case. The testing gets “squished” because coding takes longer than expected, and because teams get into a code-and-fix cycle at the end. (Crispin and Gregory 2009, 12)

From a documentation perspective, more challenges and opportunities arise. The challenge the documentation topic presents in the Agile methodology is determining the best way to implement it. One source calls the Agile manifesto reference to documentation “nothing more than an attempt to legitimize hacker behavior” (Rakitin 2001, 4). Another one states that the developer of test driven development, Kent Beck believes, “…well-written code is its own documentation. In other words, the code should be so clear and well-written that it needs so [no] extra documentation in the traditional sense” (Hoda, Noble, and Marshall 2010, 1). As is evident, opinions differ dramatically on this topic. So, while the manifesto suggests that working software is more valuable than documentation, one source suggests developers have construed this to mean “We want to spend all our time coding. Remember, real programmers don’t write documentation” (Rakitin 2001, 4). This source goes on to suggest that developers then use this suggestion that documentation is not valuable as leverage with management, stating:

> Processes, tools, documentation, and plans are useful. But when push comes to shove—and it usually does—something must give, and we need to be clear about what stays and what gives. The issue here is that when push comes to shove, management caves into the pressure out of fear. Some hackers have management convinced that process, tools, documentation, and contracts—which are, by the way, the norm in every other industry—cause the pushing and shoving. (Rakitin 2001, 4)

Conversely, other practitioners suggest that the lack of definition and standard in the Agile methodology proves that "…Agile methods don't shun documentation per se, but requires the practitioner to justify its existence" (Hoda, Noble, and Marshall 2010, 1). This suggestion forces the project members to define the documentation requirement that results in value added for the project as a whole. Note this statement places the burden on the project to define the requirement. In determining this balance, this begs the questions: How effective overall is a project that executes complete documentation? And how effective is written communication? Figures 27 and 28 pictorially address these questions.

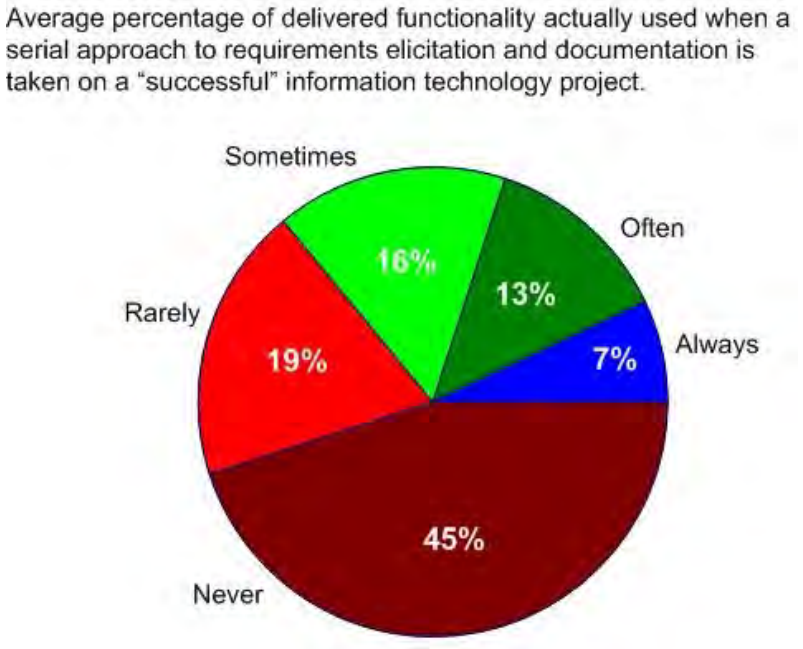

Figure 27. Success Percentage for Projects Incorporating Comprehensive Documentation (from Scott Ambler and Associates n.d.-b)

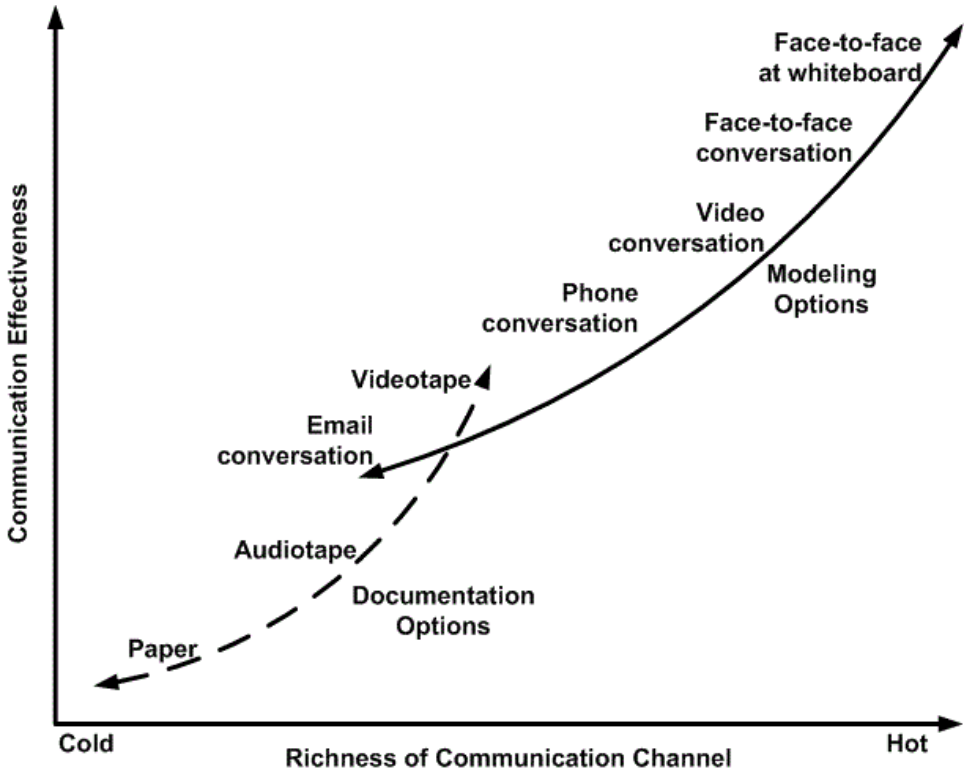

Figure 28. Effectiveness of Communication Methods (from Scott Ambler and Associates n.d.-b)

These figures show that full, robust documentation does not equate to success and that paper is the poorest communication method giving more credibility to a project being very discerning as it determines what and how to document. Furthermore, Andreas Rüping in his book *Agile Documentation*: *A Pattern guide to Producing Lightweight Documents for Software Projects*, states:

> My view is that a light-but-sufficient approach is favourable for two reasons. First, such an approach prevents the project team from expending unnecessarily large effort on documentation. Second, light-but-sufficient documentation is more accessible, and therefore more useful, for a team than voluminous documentation. (Rüping 2003, 3)

He goes on to suggest that the more documentation there is, the less useful it becomes, Figure 29.

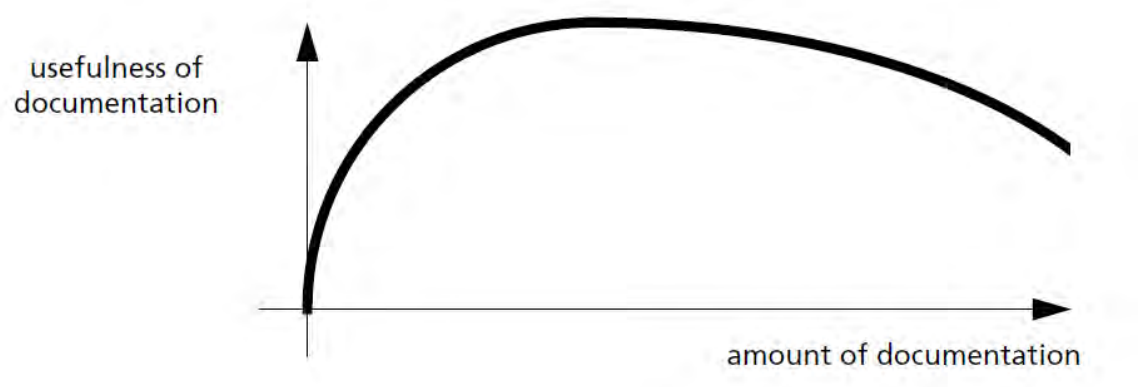

Figure 29. Usefulness of Documentation (from Rüping 2003, 4)

To combat this potential, this author suggests that any documentation should have a purpose and focus on quality. He states:

> People sometimes get the impression that, in an agile context, not only is lightweight documentation given preference over comprehensive documentation, but also that quality isn't so important. I think this is a misconception, and clearly I disagree. If you decide that a document is necessary, then it must have a purpose, otherwise you wouldn't make the decision to create it. But to fulfil that purpose, a certain quality is essential. (Rüping 2003, 4)

Agile projects should attempt focus on documentation that suits the project and the requirements. One project may have requirements that warrant a simple conversation between developers in "…informal discussion…" vice another where "…documentation may be the project's goal." (Rüping 2003, 28)

2. Agile Team Responsibilities

As with the other project facets covered in this thesis, there are many persons involved in the testing and documentation aspects of an Agile project. From a test perspective, one of the first questions to resolve is whether or not the test team staff should be integrated into the development team or separated and independent. One source lists some possible dangers to the integrated approach:

- "It is important to have that independent check and audit role.

- The team can provide an unbiased and outside view relating to the quality of the product.
- If testers work too closely with developers, they will start to think like developers and lose their customer viewpoint.
- If the testers and developers report to the same person, there is a danger that the priority becomes delivering any code rather than delivering tested code." (Crispin and Gregory 2009, 60)

This set of authors goes on to suggest that while these potential risks exist, detaching the test team can cause other unwanted consequences (Crispin and Gregory 2009, 60). This separation can naturally generate an "us versus them" mentality that can lead to the various team factions focusing on disparate objectives. Integrating the testers with the development team has the opportunity to lead to better communication between factions thereby improving the quality of the product (Crispin and Gregory 2009, 61). This being the case, the authors write:

> We don't believe that integrating the testers with the project teams prevents testers from doing their jobs well. In fact, testers on agile teams feel very strongly about their role as customer advocate and also feel they can influence the rest of the team in quality thinking. (Crispin and Gregory 2009, 60)

This integrated compared to separated approach is pictorially depicted in Figure 30.

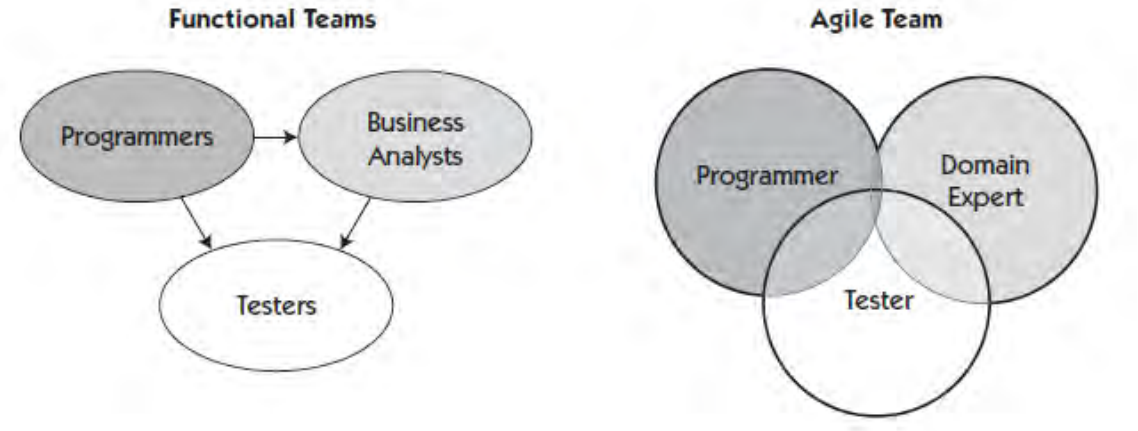

Figure 30. Independent versus Integrated Test Teams (from Crispin and Gregory 2009, 64)

Another source takes this integrated team concept even further, suggesting that "In agile projects, everyone on the team fully tests their own work. So, professional testers add value not through more testing, but by writing some of the developers' tests, thus freeing them to code more new features" (Talby et al., 2006, 33). This integration of professional testers, the authors warn, has the potential to cause one of the issues listed above with regard to "contamination" of the test discipline due to lack of independence from the development team; as well as, leading to testers not focusing on the system specification, but rather on developer based direction (Talby et al., 2006, 33). While the source does not dispute these concerns, the authors argue the "…alternative is better overall" (Talby et al., 2006, 33). For this assertion, the authors cite the fact that the more focus there is on testing, the more defects that can be uncovered better the system will be (Talby et al., 2006, 33). From a specific case study, the number of test of steps executed by the entire test team versus only the testers depicts this point in Figure 31.

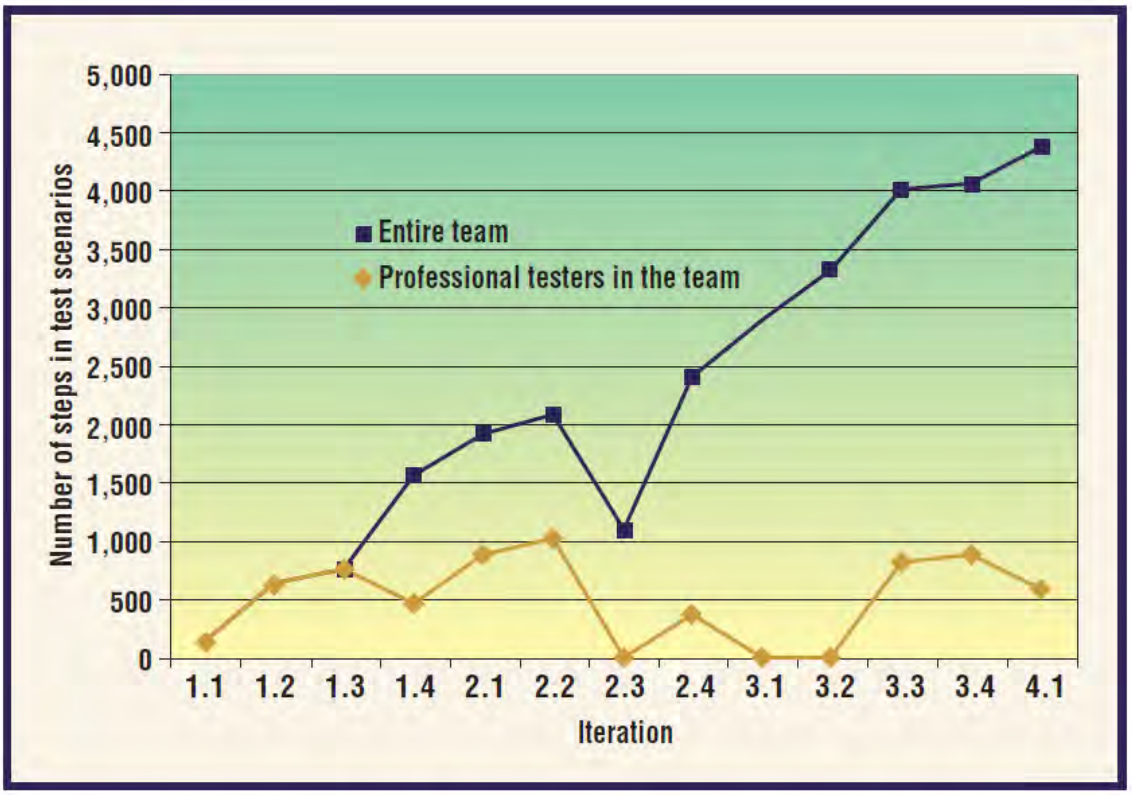

Figure 31. Entire Development Team versus Tester Team Test Steps Executed (from Talby et al., 2006, 33)

To better define the relationship between the tester and development team, another author shows that various triads are formed to support the development and execution of tests. At times, this triad may consist of the developer, a system architect, and the tester to help generate tests cooperatively. However, in other cases, it may involve the developer, database architect, and the tester. Whatever ad hoc team that is required, the goal is to "…getting an end-to-end system working soon after the start of the project" as this allows the feedback loop from the customer that is essential in validating the system (Pugh 2011, 167–169).

From the documentation perspective, there are many opinions. Some believe that developers should practice code development such that it is self-documenting to assist in future evaluation (Hoda, Noble, and Marshall 2010, 1), but others suggest that "…documenting software at the code level is foolish" (Selic 2009, 12); while yet another source based on a survey of 76 professional software maintainers voted that source code combined with comments is the most important project artifact (de Souza, Anquetil, and de Oliveira 2006, 38–40). In the same way, abstracted document artifacts are also debated. One source suggests that there should be many contributors, but one author. This is defined by Andreas Rüping as he writes in regards to this topic, "If a large number of people are responsible for one task, it is likely that the task will not be done at all—everybody will think that someone else is in charge" (Rüping 2003, 164). He goes on to suggest, that this person should be motivated to work on this task and, ideally, be a member of the project team. It is also important, the author continues, that he/she solicit inputs from fellow project team members and ensures reviews are executed and resulting data incorporated (Rüping 2003, 164–165). However, while this may be the ideal recommendation, in a study performed by Christoph Johann Stettina, Werner Heijstek, and Tor Erlend Fægri, documentation fell to a single individual—but not because they were uniquely positioned or motivated, but rather because they were the least skilled developer (2012, 34). This result turned a team of what should be generalists performing all tasks on a project team, to specialists as shown in Figure 32.

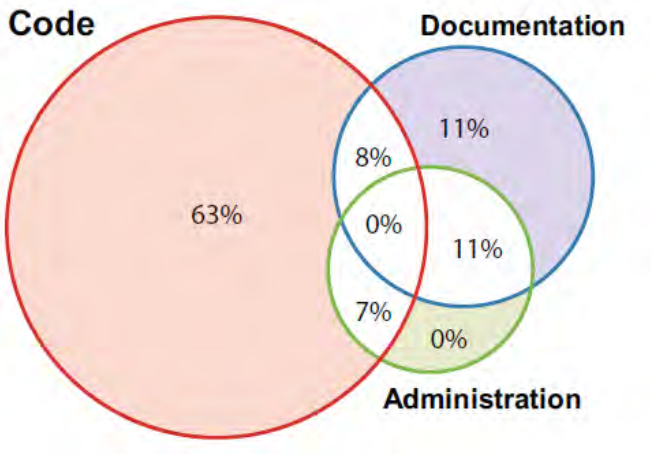

Figure 32. Distribution of Team Roles (from Stettina, Heijstek, and Fægri 2012, 35)

The authors write:

> This is very far from the 'agile ideal' where team members are generalists and avoid specialized roles. The circles would then have much greater overlap. Agile team members do not have specific roles, instead each team member has a variety of roles. This encourages socialization within the team and is an important enabler for knowledge creation. (Stettina, Heijstek, and Fægri 2012, 37)

As expected, the results of this study suggest that "time pressure" played a key factor one person being saddled with this responsibility. To avoid this, the author's state, "Explicit rotation of roles and more effective documentation tools may create the necessary stimulus for sharing the documentation tasks."

3. Process Methods

The Agile approach to testing is fundamentally different from the traditional method. Figure 33 provides one illustration of this notion. This author suggests that the traditional method of testing following a "waterfall" approach is a "code and fix" method focused on "finding bugs," whereas the Agile approach attempts to "prevent bugs" (Brown 2014). This requires a commitment from the developers to execute many more unit tests making up the base of the pyramid. This approach also requires the implementation of the "The Thin UI [user interface]" or "The Humble Dialog Box" development model (Brown 2014). This design approach ensures the application's UI

layer contains the most modest level of logic, leaving that function to lower layers of the application (e.g., business logic layer). This allows for automated tests to be executed to the greatest degree and limiting UI testing either via manual means or automated using user recorded steps which are prone to error and increased maintenance. This is pictorially illustrated in the UI tests that are the tip of the Agile pyramid and the largest segment of the traditional one (Brown 2014).

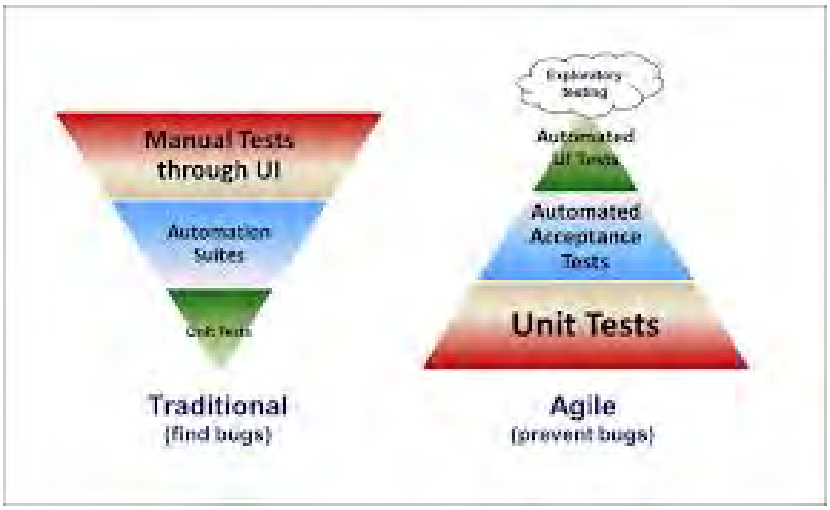

Figure 33. Traditional and Agile Testing Pyramids (from Brown 2014)

This automation offers greater code coverage, easier bug analysis and resolution, higher quality assurance, and constant feedback. An important element unique to Agile not to be lost in this discussion is the notion of acceptance tests. As defined by one author, acceptance tests are "…customer-defined tests created prior to implementation. You can use many of these acceptance tests at multiple levels both as implementation validation tests and design verification tests" (Pugh 2011, 210). These are tests that are based on customer defined acceptance criteria, and can be run from the customer perspective to ultimately ensure valid implementation of the system (Pugh 2011, 211).

One process method that can be used in developing to acceptance criteria to help encourage and enforce the Agile model is test driven development (TDD). TDD is designed considering lean principles trying to "…reduce waste in process" (Pugh 2011, 20). This author elaborates on this notion stating, "Creating acceptance tests up front

reduces waste by decreasing the loopbacks from testing back to coding" (Pugh 2011, 20). He goes on to suggest that this in turn builds system integrity as the tests are run early and repeatedly, rather than waiting until the end of the development to begin the process. TDD is a strategy in which a developer prepares a test for desired functionality, executes that test to validate failure, and then composes code that resolves the requirement (Pugh 2011, 20). Then, the developer moves to the next requirement doing the same operation, and writes steps that mixes the two together creating an integration test (Crispin and Gregory 2009, 5). At the most atomic level, the tests prepared by a developer on a single set of code (e.g., method or class) would be considered a unit test; and, one author extends the TDD definition to include automating these unit tests as well (Janzen and Saiedian 2005, 43). These tests can then be executed individually or in bulk in an automated fashion saving time compared to a manual test process (Janzen and Saiedian 2005, 43). This practice fundamentally changes the focus of the developer. Traditionally the code is written to satisfy a requirement and subsequently tested to validate it. TDD forces the developer to write code that satisfies a test which promotes the notion "…that test can aid in deciding what program code to write and what the program's interface should look like…" (Janzen and Saiedian 2005, 44). This set of automated tests can be used as the development progresses to provide quick response on whether a new implementation causes a test failure. However, if that does occur, the developer is notified quickly and able to make an update to positively affect the failure while most familiar with the unit Janzen and Saiedian 2005, 44). According to one source:

> The process of writing tests first helps programmers design their code well. These tests let the programmers confidently write code to deliver a story's features without worrying about making unintended changes to the system. They can verify that their design and architecture decisions are appropriate (Crispin and Gregory 2009, 99).

Studies have also shown marked improvement using this strategy, including (Janzen and Saiedian 2005, 48–49):

- TDD passed 18 percent to 50 percent more external tests
- Less time spent debugging code developed with TDD
- 40 percent to 54 percent reduction in defects

- Improved, minimal, and up to 16 percent negative impact to programmer productivity (although the authors note the latter was the result of the control group composing much less tests)

Testers often assist developers in working on TDD supporting the developer by ensuring the code is tested from every perspective, including considering other tests from other developers as well (Crispin and Gregory 2009, 111). In addition, since developers are performing these unit test and finding programming defects, this frees up the tester to consider the broader tests that the developer may not have considered from a more abstracted perspective. This allows for a tester to also engage in exploratory testing providing further code coverage and uncovering unsatisfied requirements. One author summarizes this concept stating, "After a development team has mastered TDD, the focus for improvement shifts from bug prevention to figuring out better ways to elicit and capture requirements before coding" (Crispin and Gregory 2009, 113).

One Agile approach to documentation is captured in Figure 34. In this pattern, the author describes that the project team should begin each document identifying exactly who the audience ("Target Readers") of the artifact will be. This initial step forces the author to consider the role and background of the audience allowing an easier opportunity to satisfy the second step of "Focused Information" (Rüping 2003, 24–25). The author explains having a well-defined focus helps keep the document "…short and concise…" allowing not only the author to spend less time in preparing it, but also the reader the ability to review the information quickly (Rüping 2003, 26). Similar to determining the understanding the audience for the document, each project should also ensure it defines "individual documentation requirements." Some projects have stringent documentation needs, while others only require much less. This attention to each document allows for definition of the necessary data required and plan effectively for its generation (Rüping 2003, 28–29).

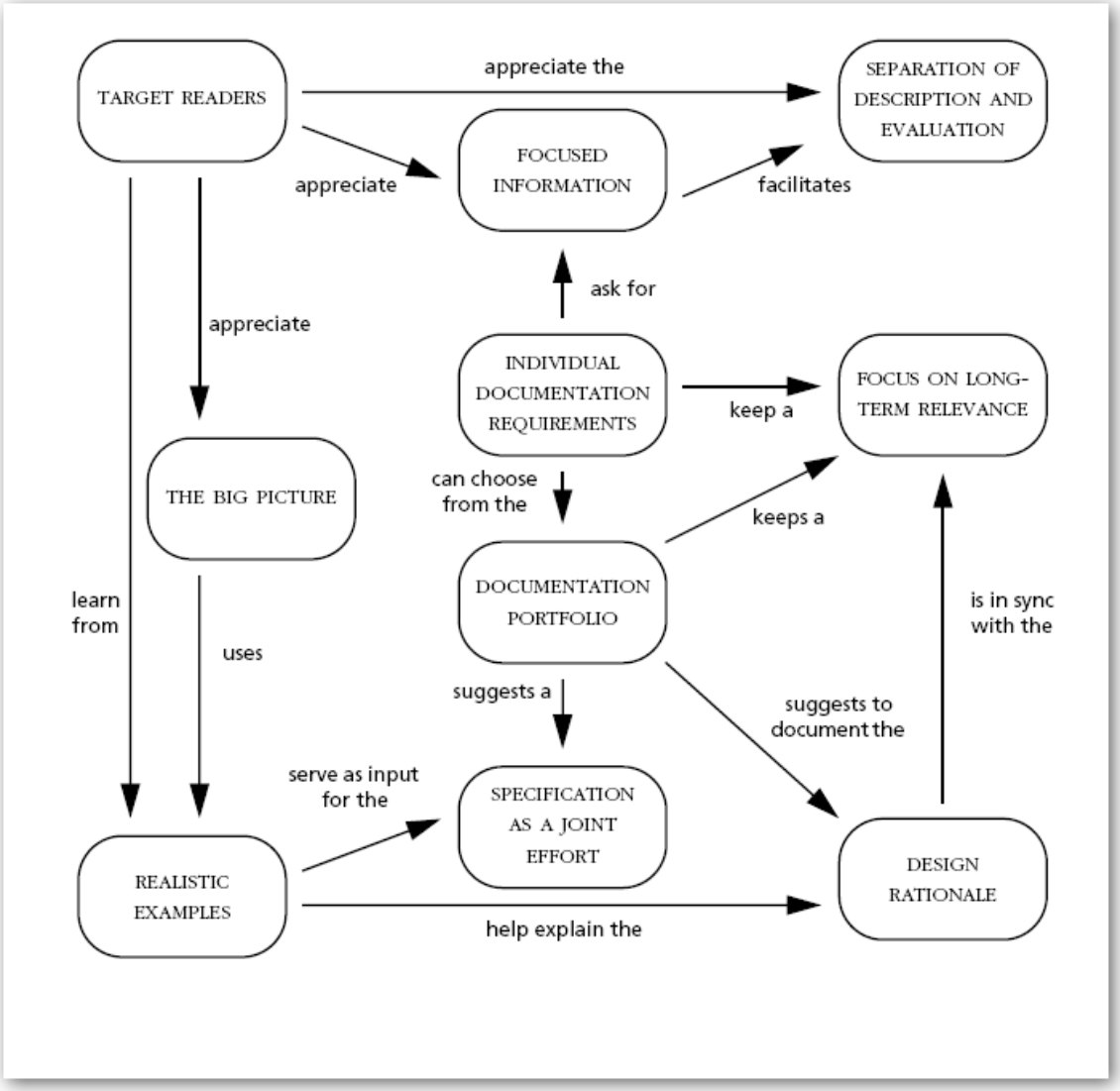

Figure 34. Agile Documentation Approach (from Rüping 2003, 23)

These "individual documentation requirements" can be selected from the "documentation portfolio" for the project. These documents assist a project in easily down selecting to the "right" set of documents for each unique project as shown in the Figure 35 example (Rüping 2003, 32).

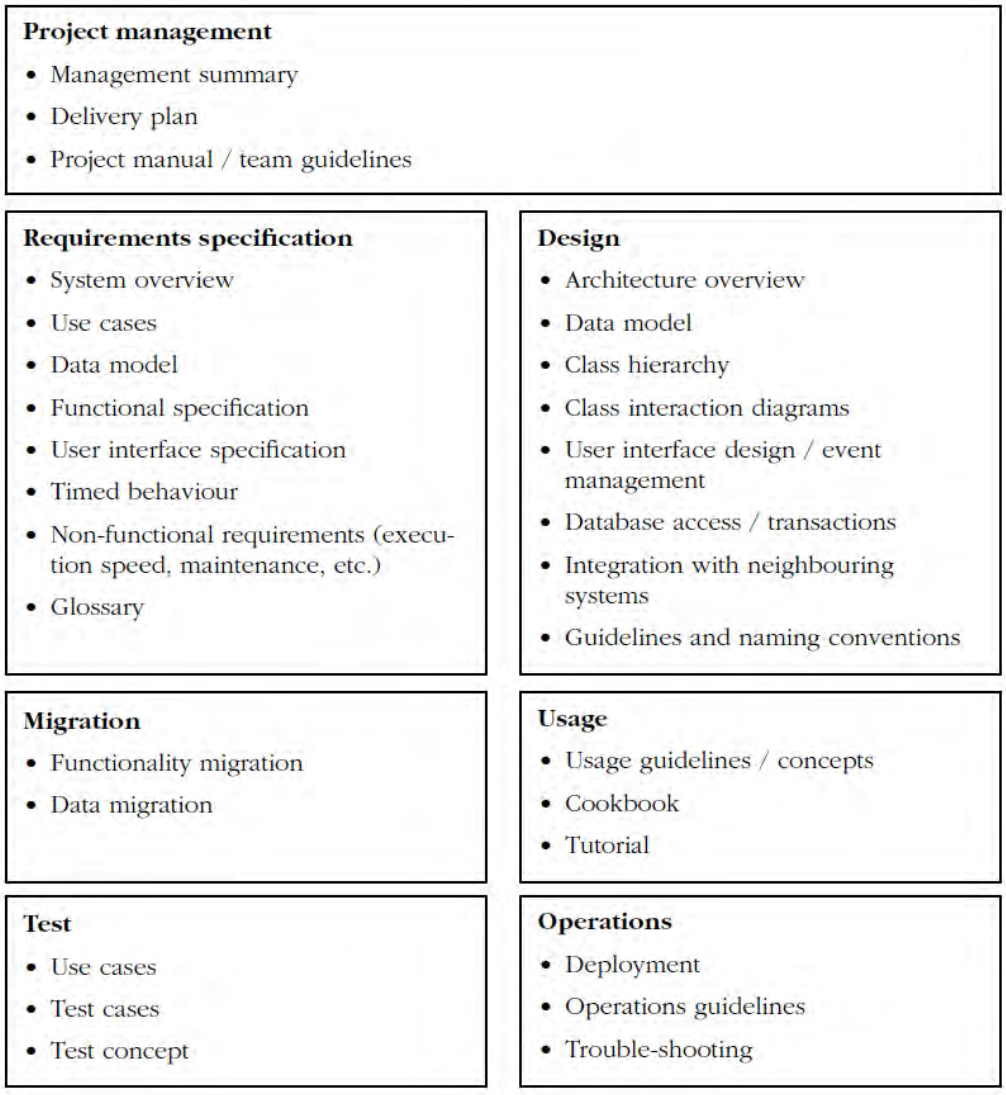

Figure 35. Project Documentation Portfolio (from Rüping 2003, 32)

Projects must also ensure that the artifacts prepared "focus on long-term relevance." Ensuring that documentation stores relevant information that will be required over time is one of the rationales for preparing documentation, consequently, the author writes, "There is much value in documentation that focuses on issues with a long-term relevance—issues that will play a role in a later project phase or in future projects" (Rüping 2003, 35).

A specification from an Agile project documentation perspective is best prepared as a joint collaboration between the development team and the customer, thus the

"Specification as a Joint Effort" step. The author highlights the fact that customer is the domain authority for the project and can help with many of the details of the specification resulting in an agreed upon set of requirements (Rüping 2003, 38). As design is performed on the system, it is important to document the "design rationale" as well as the definition. This is important to future maintenance efforts as, "The rationale behind a design is what is useful for team members who need to understand the internals of the software, perhaps because they have to maintain, extend or improve it…" (Rüping 2003, 40). Although the design details are extremely important, the "big picture" of the system in question is also crucial. The document(s) outlining this data should be relatively short and provide direction to other more detailed documents, but overall provide a general overview of the project concept (Rüping 2003, 40–41). Regardless of which document the author is preparing, it is important that there is a clear "Separation of Description and Evaluation." The author recognizes the importance of both elements in a document but states that "It isn't good style to try to influence readers by confusing description and judgement—readers might doubt the contents of a document that seems to be suggestive" (Rüping 2003, 43). In the same way, documents can gain credibility from incorporating "realistic examples" throughout the text. These impart the readers with an increased sense of confidence in the artifacts. For this reason, the author recommends liberal use of use cases and scenarios throughout a specification or other document (Rüping 2003, 45).

C. CHAPTER SUMMARY

Although the Agile Manifesto discriminates against documentation, testing and defining a manner of recording the resulting design and other key project elements can be as important to the Agile development methodology as any other development process. From a testing perspective, Agile recommends perform test activities from the beginning of a project with a special emphasis on unit tests. While this differs from a traditional approach, it does offer many advantages. On the other hand, documentation must be made a priority by the product owner for it to be effective. The Agile viewpoint is to avoid waste, and much documentation is viewed from this perspective.

All project team members are involved in the testing and documentation processes, it is important that the roles of tester and author are defined and understood. The tester role does not have to be detached from the project team, but may be an integrated member. In addition, it is important to note that there is an expectation on an Agile development project that the bulk of the tests are unit tests executed by the development team (see Figures 31 and 33). On the documentation side, developers are responsible for writing code that is self-documenting, while for documents all project members need to provide assistance while there should be a single author responsible for the document outcome.

TDD has emerged as a popular development strategy for testing. Generating the test definition before the development helps the developer ensure passing the resulting evaluation, in some ways generating a completely different and improved development strategy. If these resulting tests are also placed into an automated test set up this can improve the resulting product, reducing defects dramatically. Documentation also incorporates an Agile approach. Applying concentrated patterns (Figure 34) to the development of project artifacts can increase the value of these documents, ensuring the information is effective and incorporates credible information.

VII. CONCLUSIONS

A. KEY POINTS AND RECOMMENDATIONS

The Agile development model is the favorite of the next wave of life cycle models and is currently assisting more projects using a formal development methodology than any others by a significant margin according to research results from Forrester Research (Figure 36 and Figure 37) (West and Grant 2010, 2).

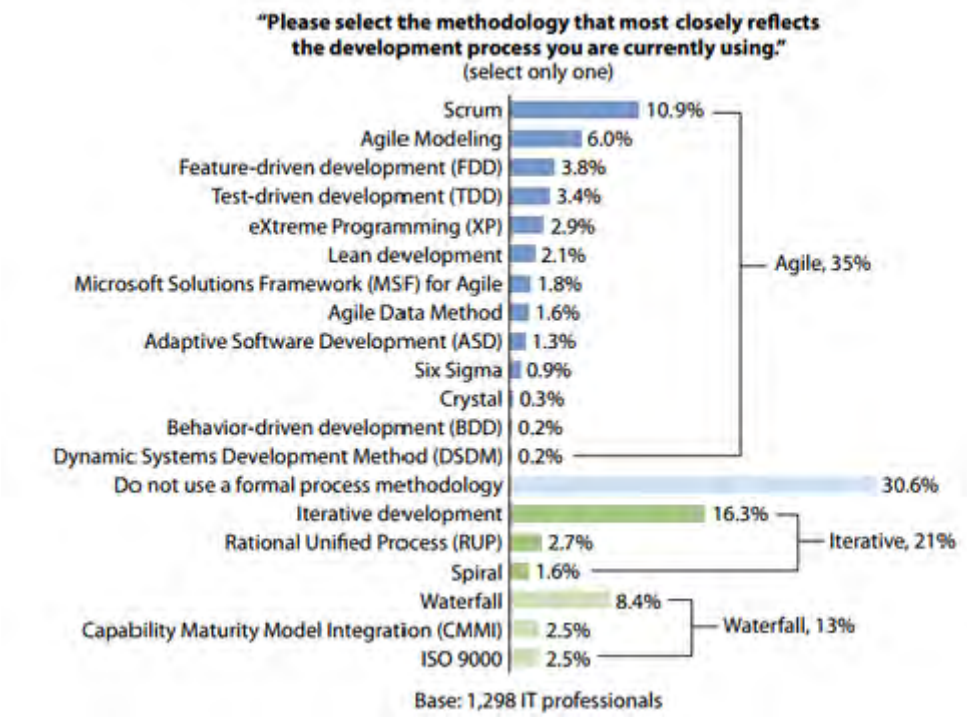

Figure 36. Development Methodology Used-1 (West and Grant 2010, 2)

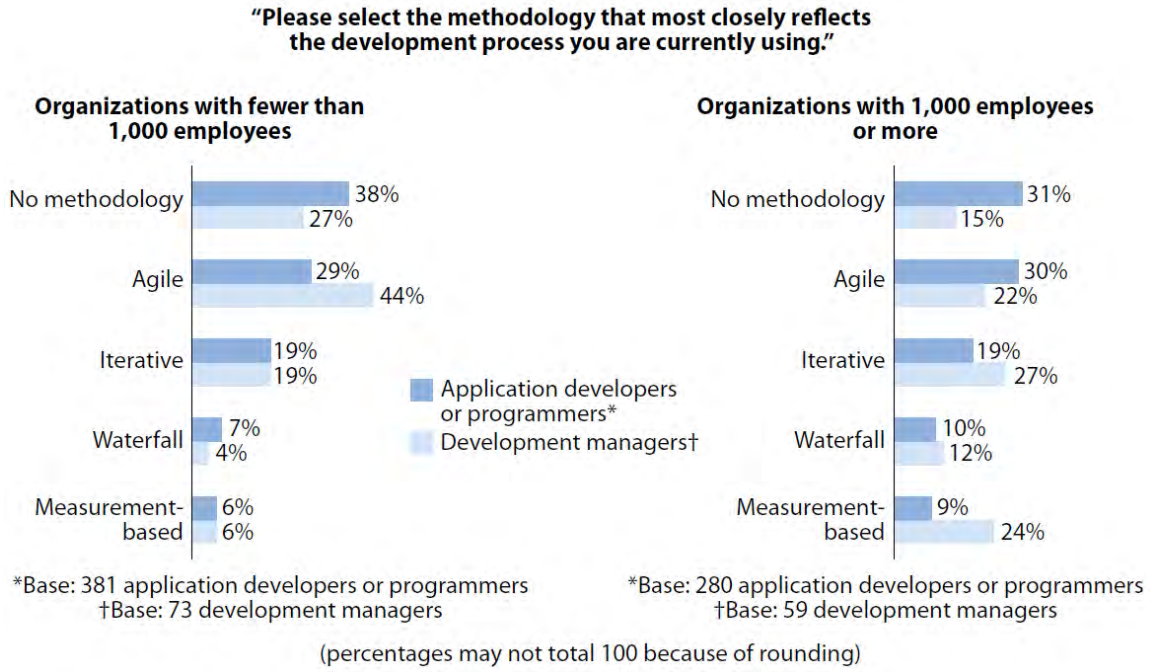

Figure 37. Development Methodology Used-2 (West and Grant 2010, 3)

In fact, according to this source, the percentage of professionals using Agile development swells to "…45% if you expand what you include in Agile's definition" (West and Grant 2010, 2). However, while Chapter I of this thesis shows that Agile provides tremendous upside in business value (Figure 1 and Figure 2), it also reveals how the methodology tends to lack discipline in project management areas. The primary objective of this thesis is to evaluate this aspect of the methodology evaluating the pros and cons of the approach, defined the roles of the Agile development team and the responsibilities each one has, and identifying processes and methods that may be used. Starting with the Agile Manifesto it is clear that much of this definition was left out purposely to force projects independently and individually to select the "right" best practices to be exercised for an individual project. The rationale for this perceived lack of definition in the model is actually a portion of the Agile models itself. Performing the "right" amount of process is the Agile challenge.

Overall, it appears from the analysis of the varied sources that the Agile methodology has many opportunities for instilling effective, robust project management

practices on any project. However, it is also clear that this direction must be made a priority and should be iteratively revisited during project execution. This starts with the stakeholders establishing its importance, being prioritized appropriately within the product backlog by the product owner, establishing focus by the development team leadership, and ultimately being owned and executed by the development team itself.

B. AREAS TO CONDUCT FURTHER RESEARCH

To further this thesis study, recommend performing a case study of both development and maintenance projects in completed Agile software developments to measure success rates and lessons learned. This analysis may also be expanded to include the application of the Agile approach on non-development tasks such as systems engineering or training development. In addition, as the product owner plays such an important role in prioritizing development work and deliverables from requirements management to documentation, further research should consider bias issues from the product owner perspective, and expand on how bias may effect successful program execution.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Abrahamsson, Pekka, Muhammad Ali Babar and Philippe Kruchten. 2010. "Agility and Architecture: Can They Coexist?" *IEEE Software* 27(2):16-22.

Agile Manifesto. 2001. "History: The Agile Manifesto." Accessed May 4, 2014. http://agilemanifesto.org/history.html.

Aravinda, Shanaka. 2012. "Week 6: Project Risk Management." *Project Management Blog*. September 2. http://shanaka90.blogspot.com/2012/09/week-6_2.html.

Bass, Len., Paul Clements, and Rick Kazman. 2003. *Software Architecture in Practice*. Boston, MA: Addison-Wesley.

Brown, R. 2014. "The Agile Testing Pyramid." Agile Coach Journal, 28 January. http://www.agilecoachjournal.com/index.php/2014-01-28/testing-2/the-agile-testing-pyramid/.

Ceschi, Martina, Alberto Sillitti, Giancario Succi, and Stefano De Panfills. 2005. "Project Management in Plan-Based and Agile Companies." *IEEE Software* 22(3):21–27.

Chin, Gary. 2004. *Agile Project Management: How to Succeed in the Face of Changing Project Requirements*. New York: AMACOM.

Cobb, Charles G. 2011. *Making Sense of Agile Project Management: Balancing Control and Agility*. Hoboken, NJ: John Wiley & Sons, Inc.

Cohn, Mike. 2012. "Agile Succeeds Three Times More Often Than Waterfall." *Mike Cohn's Blog*. February 13. http://www.mountaingoatsoftware.com/blog/agile-succeeds-three-times-more-often-than-waterfall.

Cohn, Mike. 2013. "Clarifying the Relationship between Definition of Done and Conditions of Satisfaction." *Mike Cohn's Blog*. August 21. http://www.mountaingoatsoftware.com/blog/clarifying-the-relationship-between-definition-of-done-and-conditions-of-sa.

Cooke, Jamie Lynn. 2010. *Agile Productivity Unleashed: Proven Approaches for Achieving Real Productivity Gains in Any Organization*. Ely, Cambridgeshire, UK: IT Governance Publishing.

Crispin, Lisa and Janet Gregory. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Boston, MA: Pearson Education, Inc.

Delgadillo, Lorena and Orlena Gotel. 2007. "Story-Wall: A Concept for Lightweight Requirements Management." Paper presented at the 15th IEEE International Requirements Engineering Conference, New Delhi, India, October 15–19.

DotNetBlocks. 2011. “Waterfall Model (SDLC) vs. Prototyping Model.” DotNetBlocks, April 11. http://www.dotnetblocks.com/post/2011/04/25/Waterfall-Model-%28SDLC%29-vs-Prototyping-Model.aspx.

Gottesdiener, Ellen. 2002. *Requirements by Collaboration*. Indianapolis, IN: Pearson.

Glazer, Hillel, Jeff Dalton, David Anderson, Michael Konrad, and Sandra Shrum 2008. *CMMI or Agile: Why Not Embrace Both!* (CMU/SEI-2008-TN-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm.

Hansenne, Rami and Allan Hibner. 2011. “Overcoming Organisational Challenges related to Agile Project Management Adoption.” MBA Thesis, Blekinge Institute of Technology, Karlskrona, Sweden.

Hoda, Rashina, James Noble, and Stuart Marshall. 2010. “How Much is Just Enough? Some Documentation Patterns on Agile Projects.” Paper presented at the 15th European Conference on Pattern Languages of Programs, Irsee Monastery, Germany, July 7–11.

Janzen, David and Hossein Saiedian. 2005. “Test-Driven Development: Concepts, Taxonomy, and Future Direction.” *Computer*, 38(9):43–50.

Klein, Mark H. 2008. “Software Architecture Technology Initiative.” Lecture presented at the Software Architecture Technology User Network (SATURN) Workshop, Pittsburgh, PA, April 30–May 1. Accessed October 10, 2014. http://resources.sei.cmu.edu/asset_files/Presentation/2008_017_001_23538.pdf.

Kruchten, Philippe. 2010. “Software Architecture and Agile Software Development—A Clash of Two Cultures?” Paper presented at the ACM/IEEE 32nd International Conference on Software Engineering, Cape Town, South Africa, 2- 8 May.

Leffingwell, Dean. 2011. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Boston, MA: Pearson.

Mikoluk, Kasia. 2013. “Agile vs. Waterfall: Evaluating the Pros and Cons.” *Udemy/ BLOG*. September 9. https://www.udemy.com/blog/agile-vs-waterfall/.

Misra, Subhas C., Uma Kumar, Vinod Kumar and Mahmud A. Shareef. 2007. “Risk Management Models in Software Engineering.” *International Journal of Process Management and Benchmarking*. 2(1): 59–70.

N-Axis Software Technologies. 2010. “Agile Scrum Methodology.” Accessed May 5, 2014. http://www.n-axis.in/methodologies-agile.php.

Paasivaara, Maria, Sandra Durasiewicz and Casper Lassenius. 2009. “Using Scrum in Distributed Agile Development: A Multiple Case Study.” Paper presented at the 2009 Fourth IEEE International Conference on Global Software Engineering, Limerick, Ireland, July 13–16.

Pugh, K. 2011. *Lean-Agile Acceptance Test-Driven Development: Better Software through Collaboration.* Boston, MA: Pearson.

Rakitin, Steven R. 2001. “Manifesto Elicits Cynicism.” *IEEE Computer* 34(12): 4.

Ruhe, Gunther and Greer, Des. 2003. “Quantitative Studies in Software Release Planning under Risk and Resource Constraints.” Paper presented at the 2003 International Symposium on Empirical Software Engineering, Rome, Italy, September 29–October 4.

Rüping, Andreas. 2003. *Agile Documentation*: *A Pattern guide to Producing Lightweight Documents for Software Projects*. West Sussex PO19 8SQ (England): John Wiley & Sons.

Schwaber, Ken. 2004. *Agile Project Management with Scrum*. Redmond, WA: Microsoft Press.

Scott Ambler and Associates. n.d.-a. “Agile Architecture: Strategies for Scaling Agile Development.” Accessed May 12, 2014. http://www.agilemodeling.com/essays/agileArchitecture.htm#TowardsAgileArchitecture.

Scott Ambler and Associates. n.d.-b. “Agile/Lean Documentation: Strategies for Agile Software Development.” Accessed July 28, 2014. http://www.agilemodeling.com/essays/agileDocumentation.htm#ProjectSuccess.

Selic, Bran. 2009. “Agile Documentation, Anyone?” *IEEE Software* 26(6): 11–12.

Smith, Preston G., and Roman Pichler. 2005. “Agile Risks/Agile Rewards.” *Software Development*. 13 (4):50–53.

de Souza, Sergio Cozzetti B., Nicolas Anquetil, and Káthia M. de Oliveira. 2006. “Which Documentation for Software Maintenance?” *Journal of the Brazilian Computer Society* 12(3): 31–44.

Standish Group International, Inc. 1995. *The Standish Group Report: CHAOS*. Accessed May 26, 2014. http://www.projectsmart.co.uk/docs/chaos-report.pdf.

Stevens, Matt, and Doug Rosenberg. 2003. *Extreme Programming Refactored: The Case Against XP*. CA: APress.

Stettina, Christoph Johann, Werner Heijstek, and Tor Erlend Fægri. 2012. "Documentation Work in Agile Teams: The Role of Documentation Formalism in Achieving a Sustainable Practice." Paper presented at the Agile Conference (AGILE) 2012, Dallas, Texas, August 13–17.

Takeuchi, Hirotaka, and Ikujiro Nonaka. 1986. "The New New Product Development Game." *Harvard Business Review* 46(1):137–146.

Talby, David, Arie Keren, Orit Hazzan, and Yael Dubinsky. 2006. "Agile Software Testing in a Large-Scale Project." *IEEE Software* 23(4):30–37.

VersionOne, 2013. "Benefits of Agile Software Development." Accessed May 5, 2014. http://www.versionone.com/Agile101/Agile-Software-Development-Benefits/.

Waterfall Model. 2014. "All About the Waterfall Model." Accessed October 4, 2014. http://www.waterfall-model.com/.

West, Dave and Tom Grant. 2010. *Agile Development: Mainstream Adoption Has Changed Agility for Application Development & Program Management Professional*. Cambridge, MA: Forrester Research, Inc.

Whitaker, Ken. 2010. *Principles of Software Development Leadership: Applying Project Management Principles to Agile Software Development Leadership*. Boston, MA: Course Technology.

Yakyma, Alex and Dean Leffingwell. 2013. "The Principles of Agile Architecture." January 2. http://scaledagileframework.com/the-principles-of-agile-architecture/.

Young, Cynick and Hitoki Terashima. 2008. "How Did We Adapt Agile Processes to Our Distributed Development?" Paper presented at the 2008 AGILE Conference, Toronto, Canada, August 4–8.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California